НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)

"___"_____2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «*Комплекс задач підтримки процесу написання віршів*»

Виконав: студент *IV* курсу, групи *ІС-61*

Хоменко Олександр Миколайович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц. к.ф.-м.н. Гавриленко Олена Валеріївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц. к.т.н. Тєлишева Тамара Олексіївна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *ст. викл. каф. ТК, к.т.н. Солдатова М.О.* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2020 р.

З А В Д А Н Н Я
на дипломний проєкт студенту

Хоменку Олександру Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Комплекс задач підтримки процесу написання віршів*»
керівник проєкту *Гавриленко Олена Валеріївна, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*7*”*травня* 2020 р. №1081-с
2. Термін подання студентом проєкту “*01*”*червня* 2020 року
3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів
5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту
5. Перелік графічного матеріалу

1. Діаграма класів

2. Діаграма послідовності

3. Діаграма компонентів

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *«13» квітня 2020 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*11.03.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*17.03.2020*	
3.	*Постановка та формалізація задачі*	*22.03.2020*	
4.	*Розробка інформаційного забезпечення*	*27.03.2020*	
5.	*Алгоритмізація задачі*	*05.04.2020*	
6.	*Обґрунтування використовуваних технічних засобів*	*19.04.2020*	
7.	*Розробка програмного забезпечення*	*24.04.2020*	
8.	*Налагодження програми*	*05.05.2020*	
9.	*Виконання графічних документів*	*15.05.2020*	
10.	*Оформлення пояснювальної записки*	*15.05.2020*	
11.	*Подання ДП на попередній захист*	*31.05.2020*	
12.	*Подання ДП на основний захист*	*01.06.2020*	
13.	*Подання ДП рецензенту*	*02.06.2020*	

Студент Олександр ХОМЕНКО

Керівник Олена ГАВРИЛЕНКО

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6128.00.000 ПЗ	Пояснювальна записка	78	
3	А4	ДП 6128.01.000 ТЗ	Технічне завдання	8	
4	А4	ДП 6128.02.000 ССВ	Схема структурна варіантів використань	1	
5	А4	ДП 6128.03.000 ССД	Схема структурна діяльності процесу роботи застосунку	1	
6	А4	ДП 6128.04.000 ССК	Схема структурна класів програмного забезпечення	1	
7	А4	ДП 6128.05.000 ССП	Схема структурна послідовності	1	
8	А4	ДП 6128.06.000 ССК	Схема структурна компонентів	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 6128 00.000.00		
Розробн.		Хоменко О.М.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Гавриленко О.В.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-61	
Н/контр.		Тєлишева Т.О.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: Комплекс задач підтримки процесу написання віршів

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з шести розділів, містить 27 рисунків, 10 таблиць, 1 додаток та 25 джерел.

Дипломний проєкт присвячений розробці комплексу задач підтримки процесу написання віршів за допомогою інструментів машинного навчання.

Метою створення системи є вдосконалення процесу генерування віршів за допомогою методів машинного навчання, враховуючи контекст, риму і метр.

У розділі загальних положень описано процес генерації текстових послідовностей, побудовано функціональну модель системи, що проектується, проаналізовані існуючі аналоги, визначено її відмінності від них. Визначено мету розробки та встановлено задачі, які необхідно вирішити.

У розділі інформаційного забезпечення надано детальний опис вхідних та вихідних даних, спроектовано базу даних для збереження результатів.

Розділ математичного забезпечення присвячений обґрунтуванню вибраного методу розв’язання задачі та опису алгоритму, на основі якого створюватиметься модель нейронної мережі.

Розділ програмного забезпечення описує засоби розробки комплексу задач, обґрунтовання архітектуру програмного забезпечення та етапи проектування. Описано специфікацію функцій.

У технологічному розділі визначено мету проведення тестування програмного продукту та описано його результати.

МАШИННЕ НАВЧАННЯ, ОБРОБКА ПРИРОДНЬОЇ МОВИ, ВІРШІ, ГЕНЕРУВАННЯ ТЕКСТІВ.

ABSTRACT

The structure and the scope of the work. The explanatory note of the diploma project consists of six sections, contains 27 figures, 10 tables, 1 appendix and 25 sources.

The diploma project is devoted to the development of a set of tasks to support the process of writing poems with the help of machine learning tools.

The purpose of the system is to improve the process of generating poems using machine learning methods, taking into account the context, rhyme and meter.

In the section of general provisions the process of generation of text sequences is described, the functional model of the projected system is constructed, the existing analogues are analyzed, its differences from them are defined. The purpose of development is defined and the tasks which need to be solved are established.

The information support section provides a detailed description of input and output data, designed a database to store results.

The section of mathematical software is devoted to the substantiation of the chosen method of solving the problem and the description of the algorithm on the basis of which the neural network model will be created.

The software section describes the tools for developing a set of tasks, substantiating the software architecture and design stages. The specification of functions is described.

The technological section defines the purpose of testing the software product and describes its results.

MACHINE LEARNING, NATURAL LANGUAGE PROCESSING, POETRY, GENERATE TEXT.

ЗМІСТ

ВСТУП **6**

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ **10**

1.1 Опис предметного середовища 10

1.1.1 Опис процесу діяльності *11*

1.1.2 Опис функціональної моделі *13*

1.2 Огляд наявних аналогів 13

1.3 Постановка задачі 16

1.3.1 Призначення розробки *16*

1.3.2 Мета та задачі розробки *16*

Висновок до розділу 17

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ **18**

2.1 Вхідні дані 18

2.2 Вихідні дані 21

2.3 Опис структури бази даних 22

3 МАТЕМАТИЧНЕ ЗАБЕЗЕЧЕННЯ **24**

3.1 Змістовна постановка задачі 24

3.2 Математична постановка задачі 25

3.3. Обґрунтування методу розв'язання 27

3.3.1 Опис прихованої марковської моделі (Hidden Markov Model) *27*

3.3.2 Опис нейронних мереж (Neural Networks) *33*

3.4 Опис методів розв'язання 39

Висновок до розділу 50

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ **51**

4.1 Засоби розробки 51

4.2 Вимоги до технічного забезпечення 54

4.2.1 Загальні вимоги *54*

4.3 Архітектура програмного забезпечення 55

4.3.1 Діаграма класів *56*

4.3.2 Діаграма послідовності *58*
4.3.3 Діаграма компонентів *60*
4.3.4 Специфікація функцій *61*
4.4 Опис звітів 62
Висновок до розділу 64
5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ **65**
5.1 Керівництво користувача 65
5.2 Випробування програмного продукту 68
5.2.1 Мета випробувань *68*
5.2.2 Загальні положення *68*
5.2.3 Результати випробувань *68*
Висновок до розділу 72
ЗАГАЛЬНІ ВИСНОВКИ **74**
ПЕРЕЛІК ПОСИЛАНЬ **77**

ВСТУП

Людство стрімко розвивається протягом останього десятиліття, особливо інформаційні технології. Одним із популярних напрямів в програмуванні є робота з даними - data science. Люди вчаться обробляти великі дані, використовувати їх для побудови гіпотез, аналізувати їх, розроблювати рішення на їх основі, аналізувати і приймати рішення, опираючись на результати аналізу.

Комп'ютер - штучний інструмент для збору і аналізу даних. Машинне навчання пришвидшило роботу вчених: аналіз, створення гіпотези, проведення експериментів. Алгоритми машинного навчання здатні розробити тисячі, якщо не мільйони гіпотез, провести мільярди експериментів та проаналізувати тисячі результатів, щоб визначити найкраще рішення. Це може зайняти сотні годин, залежно від обчислювальної потужності комп'ютера [3].

Робота з мовою, особливо з текстами є одним із популярних напрямків в data science, який стрімко розвивається. Основна ідея цього напрямку – це навчити комп’ютер людської мови. Цей процес є тяжким: комп’ютер не розуміє слів, він оперує числами. Тому за допомогою математики потрібно представити слова і їх зв’язок один з одним і врахувати правила мови.

Вчені досліджували генерування текстових послідовностей: будувати граматично правильно речення, зберігати у них контекст (одну тематику), дотримання правилих пунктуації. Вони застосовували різні статистичні методи (нграми), марківські процеси і нейронні мережі. Останній з підходів виявився найкращим. Були розроблені різні модифікації моделей для вирішення специфічних задач роботи з текстом, адже кожна з них вимагала унікального підходу. Ця сфера почала активно розвиватися і використовуватися у бізнесі, що спричинило подальше дослідження,

створення нових методів для роботи з текстом, покращення існуючих і дослідження нових можливостей штучного інтелекту.

Людям завжди було цікаво, чи зможе комп'ютер навчитися відображати і передавати емоції. Одним із запропонованих варіантів стали вірші. Але комп'ютер оперує числами і не може думати різносторонньо, він не виходить за межі логіки, якої йому задала людина. Крім того вірші накладають ряд умов при написанні: контекст – повинна виконуватися одна тематика чи ідея; рима – останні слова рядків повинні римуватися; метрика – однакова структура вірша, чергування наголошених і ненеаголошених складів. Це стало викликом для ентузіастів почати дослідження у цій галузі.

За останнє десятиліття було запропоновано безліч рішень цієї задачі, в основу яких лягли нейронні мережі. Перед системами генерування віршів постає не тільки проблема структурування інформації, а й у тому, аби знайти спосіб представлення вхідних даних для подальшого аналізу, створення словника та проведення генерування віршів, контролюючи контекст, риму і структуру вірша.

Одним із етапів для генерацій віршів є збір та попередня обробка даних (текстів/віршів) та їх дослідження і аналіз. Багато систем для генерування віршів розраховані на те, що інформація вже представлена в зручній для дослідження формі. Однак, у більшості випадків, інформація з електронних джерел зберігається у вигляді неструктурованих документів. Таким чином виникає проблема знаходження інформації саме з неструктурованих текстів. Тому вченими було розроблено методи обробки текстів для попердньої їх обробки і первинного аналізу [4].

Оскільки, більшість систем генерує вірші, враховуючи вибрану тематику, структуру вірша і введені слова, вірші втрачають свою унікальність. Ці системи виривають слова і словосполучення з вхідних віршів і вставляють у результуючий, тому часто у них зустрічаються

некоректні рими і втрачається послідовність подій, головна тема і ідея. Вони генерують структурно правильні вірші, дотримуючись всіх правил, але вірш іноді важко прочитати вголос, це зумовлено неправильним підбором послідовності слів. Ці системи використовують сатистичні правила для підбору слів для вставки у вірш.

Генерування віршів стало захоплюючим напрямком в data science, оскільки цей підход можна використовувати в музиці для генерування текстів пісень. Адже пісні є схожими за структурою на вірші. Таким чином, методи обробки вхідних текстів, є важливими для того, щоб структуризувати дані і виявити ключові слова для контексту. Правильно підготовані дані подаються на вхід нейронній мережі, яка передбачить слова відповідно до контекста і місця в строфі. Структура нейронних мереж [5] дозволяє встановити залежності між слова, задовільняючи усі критерії.

На сьогоднішній день зростає кількість систем для підбору рим і генерування віршів. Це в свою чергу призвело до покращення методів для роботи з текстом і покращення алгоритмів генерування віршів.

Зростання інформації, оптимізація алгоритмів і зростання обчислювальної потужності комп’ютерів дозволяють використовувати сучасні методи для вирішення задачі класифікації.

Процес написання віршів займає досить багато часу, адже для вирішення цієї задачі необхідно не тільки підготувати вірші, а й сруктуризувати, проаналізувати їх, зрозуміти контекст і риму. Коли ж мова велику кількість даних, то часто виникають проблеми із недостачею людських і обчислювальних ресурсів. Навчання нейронних мереж є тривалим процесом і вимагає великих обчислювальних ресурсів.

Написання віршів без використання автоматизованої системи полягає у тому, аби вручну підбирати риму, контролювати метрику і слідкувати за контекстом. Кількість часу для вирішення цієї задачі значно зростає, а

сучасні інтсрументи у відкритому доступі не надають бажаного результатів. Саме тому виникає необхідність у створенні системи написання віршів для спрощення підбору рим і можливого підбору строф для вірша.

Дипломний проект присвячений розробці комплексу задач підтримки процесу написання віршів.

Практичне значення одержаних результатів

Розроблено систему генерування віршів методами природньої обробки мови та машинного навчання. Розроблений застосунок можна використовувати для підбору рими, написання тематичних віршів, рекламних слоганів чи привітань.

Публікації

Результати роботи були опубліковані у матеріалах:

1) третя Всеукраїнська Науково-Практична конференція молодих вчених та студентів «Інформаційні системи та технології управління» [1].
2) четверта Всеукраїнська Науково-Практична конференція молодих вчених та студентів «Інформаційні системи та технології управління» [2].

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

З розвитком машинного навчання збільшується інтерес до дослідження і створення нових методів штучного інтелекту. Для людини стало цікаво дізнатися, чи можливо навчити комп’ютер думати і приймати рішення як справжній людині. При цьому зростання нових теорій і одночасне зростання доступної обчислювальної потужності комп’ютерів дозволяють використовувати сучасні методи для вирішення цієї задачі. Одним із підходів є вираження думків машини через текст.

Процес створення тексту і віршів займає досить багато часу, адже для вирішення цієї задачі необхідно не тільки створити модель нейронної мережі, а й підготувати дані для навчання і провести тренування моделі. Коли ж мова йде про великі набори текстових даних, то часто виникають проблеми із недостачею обчислювальної потужності і зростанням часу тренування моделі.

У дипломній роботі пропонується рішення комплексу задач підтримки процесу написання віршів.

Автоматизоване написання віршів розглядається як інструмент, який спрощує процес створення віршів, підбор рим і римованих строф для продовження думки у вірші. Загалом, написання віршів відіграє важливу роль у відображенні емоцій через текст. Але з часом важко придумати нові рими і комбінації слів, які добре розкривають зміст вірша.

Пропонується розв’язати задачу створення віршів методами машинного навчання, використовуючи методи обробки природньої мови.

1.1.1 Опис процесу діяльності

Для того щоб вирішити задачу генерації віршів необхідно виконати наступні етапи:

- створити словник слів, проаналізувати вживання слів у різних контекстах;
- підбір коректних рим;
- слідкувати за метрикою у вірші.

Для того аби автоматизувати цей процес необхідно розв’язати задачу машинного навчання. Задачі машинного навчання можна поділити на 2 типи: навчання з учителем (supervised learning), навчання без учителя (unsupervised learning), навчання наполовину з вчителем (Semi-supervised Learning) та посилене навчання (Reinforcement Learning) [6].

При навчанні з учителем машина знає результати роботу алгоритму ще до того як почне працювати з ним або вивчати його. Машина генерує відповідь і порівнює її з очікуваним результатом і потім коригує свої параметри. При навчанні без учителя машина сама намагається зрозуміти суть алгоритму і намагається правильно передбачити результат. Дані поділяються на навчальні та тестові. Після того як модель навчилася вона звірєя свої результати з тестовими і коригує параметри [7]. Було вибрано вирішувати цю задачу саме методом без учителя, адже необхідно згенерувати вірш, а кожен з них є унікальним і неможливо точно визначити якість вірша за допомогою парметрів. Його можна оцінити тільки суб’єктивно. Для вирішення будь-якої задачі машинного навчання необхідно мати початковий набір даних (дата сет). В процесі навчання система працює з віршами, коригує параметри між нейронами і запав’ятовує вживання слів у різних контекстах і запам’ятовує порядок їх у строфі.

У кінці автоматизації ми отримаємо систему, що дозволятиме користувачеві сворювати вірші на основі введеного рядка вірша. Модель можна донавчити на додаткових даних і потім знову залити на хост-сервер.

Розглянемо структурну схему процесу діяльності, що наведена у частині графічного матеріалу.

Опишемо процес діяльності генерації віршів.

Спочатку підготуємо модель нейронної мережі, проведемо тренування.

Важливим етапом є підготовка текстових даних для тренування. Заважимо, що цей етап займає багато часу, потрібно зібрати вірші з різних електронних ресурсів і розмітити дані для тренування.

На наступному етапі відбувається створення і тренування нейронної мережі. Цей етап прихований від користувача і він виконується у системі на сервері. Після завершення тренування необхідні параметри записуються у спеціальний файл. За необхідності можна продовжити навчання, завантаживши ваги.

На наступному етапі відбувається надання доступу до моделі користувачеві. Файли з даними для настроювання мережі, зберігаються на сервері.

Відкривши сайт, користувач має можливість використати готову модель для створення віршів. На цьому етапі користувач має можливість створити власний вірш, вівши рядок і програма підбире римовані строфи. Зауважимо, що вхідний рядок повинен містити лише англійські літери, цифри і знаки пунктуації. Якщо формат не вірний, користувач отримає повідомлення про помилку і потрібно буде ввести коректну строфу.

На наступному етапі користувач має можливість спостерігати як модель підбирає слова для результуючого вірша.

Також користувачеві пропонується продовження його строфи – згенерований вірш. Користувач може його скопіювати або спробувати згенерувати ще раз. Після цього робота системи завершується.

1.1.2 Опис функціональної моделі

Опишемо функціональну модель системи за допомогою структурної схеми варіантів використання, що наведена у частині графічного матеріалу.

Розглянемо, які функції може виконувати користувач в рамках системи генерації віршів:

– генерувати вірші;
– підбирати рими, враховуючи контекст;
– тренувати модель нейронної мережі;
– підготовлювати дані для навчання.

Для того аби забезпечити користувача вищенаведеними функціями, система має виконувати такі функції:

– створення дата сету;
– тренування моделі локально;
– підбір рими відповідно до контексту;
– генерування віршів.

1.2 Огляд наявних аналогів

Проаналізуємо наявні аналоги даної системи. Детальний опис аналогів представлено у таблиці 1.1.

Таблиця 1.1 - Огляд наявних аналогів

Назва	Алгоритми	Технології	Суть задачі	Дата сет	Посилання
Neural Poetry: Learning to Generate Poems using Syllables	LSTM, Syllable-based language model	Python 3.7, Tensorflow 1.4, Numpy, Pandas, nltk.	Generating poetry	PAISA(Dante's tercets)	https://arxiv.org/pdf/1908.08861.pdf

Продовження таблиці 1.1

Назва	Алгоритми	Технології	Суть задачі	Дата сет	Посилання
Deep-speare: A joint neural model of poetic language, meter and rhyme	Char-LSTM, Attention module, Word-LSTM, LSTM, joint architecture	Python 3.7, Tensorflow 1.4, Numpy, Pandas, nltk.	Generate sonnets	Project Gutenberg	https://arxiv.org/pdf/1807.03491.pdf
Generating Chinese Classical Poems with RNN Encoder-Decoder	RNN Encoder-Decoder, bidirectional RNN with attention mechanism	Python 3.7, Tensorflow 1.4, Numpy, Pandas, nltk.	Generate Chinese classical poetry	398,391 poems from Tang Dynasty	https://arxiv.org/ftp/arxiv/papers/1604/1604.01537.pdf
GPT-2 Neural Network Poetry	GPT-2, Transformer, Attention mechanism	Python 3.7, Tensorflow 2.0, Numpy, Pandas, nltk..	Generating English poetry	Gutenberg-poetry	https://www.gwern.net/GPT-2
Rhyme With AI	BERT	Python 3.7, Tensorflow 2.0	Generating poetry	Reddit, Wikipedia, Project Gutenberg	https://devpost.com/software/rhyme-with-ai
GPT-based Generation for Classical Chinese Poetry	GPT, BERT	Python 3.7, Tensorflow 2.0, Numpy, Pandas, nltk.	Generate Chinese classical poetry	Chinese classical poetry	https://arxiv.org/pdf/1907.00151.pdf

Розглянемо особливості кожного програмного продукту, що був приведений у таблиці 1.1.

Перший програмний продукт базується на рекурентній нейронній мережі, в основі якої лежить мовна модель - склади. Це дозволяє генерувати вірші італійською мовою. На вхід нейронній мережі подаються складі, а на виході отримуємо склади, які формують слова. Вона генерує терцети не лише з належною формою, але й нагадує стиль обраного автора.

Deep-speare поєднує три моделі, щоб генерувати сонети англійською

мовою. Об’єднана модель складається з мовної моделі, моделі, яка вивчає метр, та моделею, яка досліджує римовані пари. Після обробки даних відбувається генерування сонета, для якого підбирається комбінація слів, які задовільняють усі моделі. Тренування відбувалося на сонетах Шекспіра.

Третя система базується на структурі RNN Encoder-Decoder [8], щоб генерувати чотирирядки китайською мовою із введеним тематичним словом. Вона може вивчити смислове значення в одному рядку, семантичну відповідність між рядками в поемі та використовує структурні, ритмічні та тональні зразки, не використовуючи жодних шаблонних обмежень. Механізм уваги може зафіксувати асоціації словосполучень в класичній китайській поезії і також може підвищити продуктивність.

GPT-2 Neural Network Poetry базується на архітектурі Transformer [9]. Головною перевагою цієї моделі є запам’ятовування контекстів, в яких може вживатися слова, збереження сюжетної лінії. Ця модель вже є натренованою на великій кількості даних і дотренованою на віршах з онлайн бібліотеки.

Четверта система базується на моделі Bert[10]. Ця модель є схожую до попередньої, але має менше шарів і натренована на менших об'ємах даних. У цій системі строфа будується по слову, випадково вибираючи кандита, модель підбирає слово, враховуючи попередній контекст.

GPT-based Generation for Classical Chinese Poetry генерує високоякісні класичні китайські вірші за допомогою натренованої моделі GPT. Модель генерує вірші правильної форми і дотримуючись тематики. Застосовується метод top k замість beam-search для створення різноманітного тексту.

Однією із головних відмінностей програмного продукту, що розробляється є те, що користувач самостійно вводить рядок, який модель аналізує і підстроюється під заданий стиль і тематику. Після цього вона генерує вірш на основі введеного рядка. Більшість програмних продуктів, які

я знайшов дозволяють користувачеві лише отримати кінцевий результат, обмеживши тільки тематикою або стилем вірша.

Програмний продукт, що розробляється дозволяє пройти процес навчання моделі самостійно або використати вже натреновану модель, отримати та оцінити вірш, ввівши рядок вірша, який необхідно продовжити. Модель врахує контекст попередніх рядків і відповідно римування. Різноманітність словника дозволяє генерувати унікальні комбінації словосполучень. Тому можна зробити висновок, що розробка даної програми є актуальною, оскільки жоден із аналогів не пропонує користувачеві змістовно згенеровані вірші, дотримуючись коректної рими і унікального стилю одночасно.

1.3 Постановка задачі

1.3.1 Призначення розробки

Призначенням розробки є створення віршів методами машинного навчання та інструментами для роботи з природньою мовою.

1.3.2 Мета та задачі розробки

Метою розробки є спрощення та вдосконалення процесу написання віршів за рахунок створення моделі нейронної мережі із застосуванням методів машинного навчання.

Завдяки цьому користувачі можуть створювати власні вірші англійською мовою. При генеруванні буде дотримуватися одна сюжетна лінія, римування і метрика вірша, базуючись на введеному рядку.

Для досягнення мети необхідно вирішити такі задачі:

- збереження контексту;
- підбір коректних рим;
- контроль метрики;

- візуалізація результатів.

Висновок до розділу

У ході написання даного розділу дипломного проекту було описано предметне середовище, яке розглядається. Було показано, що люди ще не досягли значних успіхів при розв’язанні задач пов’язаних із генеруванням віршів, а зростання кількості нових алгоритмів та одночасне зростання доступної обчислювальної потужності комп’ютерів дозволяють покращити процес написання віршів.

Окрім цього було описано процес діяльності, який покращується, визначено як покроково розв’язати поставлену задачу, наведено коротку характеристику підходу, яким пропонується вирішувати поставлену задачу. Було побудовано діаграму діяльності та наведено її опис.

Також був описаний функціонал системи та визначено функції користувача. Був проведений аналіз наявних існуючих аналогів та визначення відмінностей, які матиме програмний продукт, що розробляється.

Сформовано призначення розробки, встановлено мету та визначено задачі, які необхідно вирішити для досягнення мети.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Для вирішенні задачі написання віршів необхідно підготувати матеріал, для навчання нейронної мережі. У нашому випадку таким матеріалом виступають підготовлені віршовані тексти, які називаються дата сетами.

Приклад матеріалів для формування дата сету для навчання моделі нейронної мережі представлено у таблиці 2.1.

Таблиця 2.1 – Приклад дата сету

Автор	Вірш
Poems 1817 BY JOHN KEATS	What first inspired a bard of old to sing Narcissus pining o'er the untainted spring? In some delicious ramble, he had found A little space, with boughs all woven round; And in the midst of all, a clearer pool Than e'er reflected in its pleasant cool, The blue sky here, and there, serenely peeping Through tendril wreaths fantastically creeping. And on the bank a lonely flower he spied, A meek and forlorn flower, with naught of pride, Drooping its beauty o'er the watery clearness, To woo its own sad image into nearness: Deaf to light Zephyrus it would not move; But still would seem to droop, to pine, to love. So while the Poet stood in this sweet spot, Some fainter gleamings o'er his fancy shot; Nor was it long ere he had told the tale Of young Narcissus, and sad Echo's bale.
RICHARD ALDINGTON	I know that the white wind loves you, Is always kissing you and turning up The white lining of your green petticoat. The sky darts through you like blue rain, And the grey rain drips on your flanks And loves you. And I have seen the moon Slip his silver penny into your pocket As you straightened your hair;

Продовження таблиці 2.1

Автор	Вірш
LAMIA By John Keats	Ah, happy Lycius!—for she was a maid More beautiful than ever twisted braid, Or sigh'd, or blush'd, or on spring-flowered lea Spread a green kirtle to the minstrelsy: A virgin purest lipp'd, yet in the lore Of love deep learned to the red heart's core: Not one hour old, yet of sciential brain To unperplex bliss from its neighbour pain; Define their pettish limits, and estrange Their points of contact, and swift counterchange; Intrigue with the specious chaos, and dispart Its most ambiguous atoms with sure art; As though in Cupid's college she had spent Sweet days a lovely graduate, still unshent, And kept his rosy terms in idle languishment.

Окрім цього, є ще інші вхідні дані – парметри нейронної мережі але вони не задаються користувачем, а визначаються програмно. Також користувач задає рядок вірша який подається на вхід натренованій моделі. Такими параметрами є:

- vocab_size (int, за замовчуванням 50257) - розмір словникового запасу моделі GPT-2. Визначає різні лексеми, які можуть бути представлені input_ids, переданими методу переходу GPT2Model;
- n_positions (int, за замовчуванням < 1024) - максимальна довжина послідовності, з якою може використовуватися ця модель;
- n_ctx (int, за замовчуванням 1024) - розмірність маски (зазвичай така ж, як n_ positions);
- n_embd (int, за замовчуванням до 768) - розмірність вкладених та прихованих станів;
- n_layer (int, за замовчуванням до 12) - кількість прихованих шарів у кодері трансформера;

- n_head (int, за замовчуванням до 12) - кількість attention heads для кожного шару уваги в Transformer encoder;
- activation_function (str, за замовчуванням 'gelu') - функція активації, вибрана з списку [«relu», «swish», «gelu», «tanh», «gelu_new»];
- resid_pdrop (float, за замовчуванням до 0,1) - ймовірність виключення для всіх повністю з'єднаних шарів у вкладках, кодері та пулері;
- embd_pdrop (int, за замовчуванням до 0,1) - коефіцієнт виключення для вбудовування;
- attn_pdrop (float, за замовчуванням до 0,1) - коефіцієнт виключення уваги;
- layer_norm_epsilon (float, за замовчуванням 1e-5) - Епсилон для використання в шарах нормалізації шару;
- inicijalizer_range (float, за замовчуванням до 16) - стандартне відхилення усіченого_normal_initializer для ініціалізації всіх ваг матриць;

Аргумент, що використовується під час підведення підсумків послідовностей. Є одним із наступних варіантів:

- ‘Last’ => приймає останній маркер прихованого стану (наприклад, XLNet);
- 'First' => приймає перший маркер прихованого стану (наприклад, Bert);
- 'Mean' => приймає середнє значення всіх прихованих токенів;
- 'Cls_index' => надає позицію маркерів тензора класифікації (GPT);
- 'Attn' => Не реалізовано зараз, використовуйте багатоголосову увагу;
- sum_use_proj (булева, необов'язкова, за замовчуванням True) - аргумент, що використовується під час зведення послідовностей. Використовується для голови з множинним вибором у GPT2DoubleHeadsModel;

- sum_activation (рядок або None, необов’язково, за замовчуванням None) - аргумент, який використовується під час підсумків підсумків Використовується для голови з множинним вибором у GPT2DoubleHeadsModel. 'Tanh' => додайте активацію tanh до виводу, Other => немає активації;
- sum_proj_to_labels (булева, необов'язкова, за замовчуванням - True) - аргумент, який використовується під час зведення послідовностей. Використовується для голови з множинним вибором у GPT2DoubleHeadsModel. Якщо вірно, проекція виводить у config.num_labels класи (в іншому випадку на схований_размер);
- Summa_first_dropout (float, необов'язково, за замовчуванням до 0,1) - Аргумент, який використовується під час зведення послідовностей Використовується для голови з множинним вибором у GPT2DoubleHeadsModel [11].

Розглянемо кожен із вищенаведених параметрів детальніше.

2.2 Вихідні дані

Після завершення тренування моделі користувач може ввести рядок згенерувати вірш. Внаслідок роботи моделі створюються вихідні дані. Вихідними даними є:

- натренова модель нейронної мережі зберігається у вигляді файлу із машинним кодом (приклад такого файлу наведено на рисунку 2.1);
- готовий вірш.

Рисунок 2.1 – Приклад машинного коду, в якому записані параметри збереженої моделі мережі

2.3 Опис структури бази даних

Розглянемо схему бази даних, що представлена на рисунку 2.2.

Poem_info

Name	Data type	Length	Precision	Not NULL?	Primary key?
id	integer			Yes	Yes
poem_data	text			No	No
create_date	date			No	No

Рисунок 2.2 – Структурна схема бази даних

Із рисунку видно, що база даних складається із однієї таблиці: інформації про вірш, який був створений. Це дозволить відслідковувати якість роботи моделі у різні етапи життєвого циклу.

Детальний опис таблиці із інформацією про модель наведено у таблиці 2.2.

Таблиця 2.2 – Детальний опис таблиці БД із загальною інформацією

Поле	Тип	Опис
id	integer <pk>	Унікальний номер запису в таблиці.
Poem_data	text	Згенерований вірш.
Create_date	date	Дата генерування вірша.

У таблиці 2.2 із загальною інформацією міститимуться такі поля: унікальний номер запису, текст вірша, дата створення вірша.

Висновок до розділу

У ході написання даного розділу були визначені вхідні та вихідні дані системи, що розробляється.

Надано детальний опис вхідних даних, що задаються розробником і користувачем.

Встановлені параметри за замовчуванням для моделі нейронної мережі. Користувач при наявності коду може їх змінити і перевчити чи довчити мережу.

Також, проаналізовано важливість використання бази даних для збереження необхідної інформації та подальший аналіз якості створених віршів. Спроектовано модель бази даних та її структуру. Наведено детальний опис атрибутів таблиці бази даних, визначено їх тип.

3 МАТЕМАТИЧНЕ ЗАБЕЗЕЧЕННЯ

3.1 Змістовна постановка задачі

Нехай ми маємо n віршів, що були зібрані з різних електронних джерел. Така інформація для зручності наводяться у таблиці 3.1.

Таблиця 3.1 – Представлення текстів

Вірш	Автор і Назва
Вірш 1	Автор 1, Назва 1
…	…
Вірш n	Автор n, Назва n

Задача полягає у тому, щоб навчити модель створювати вірші, враховуючи значення слів у контексті, римування пар слів і метрики. Тобто створювати унікальні за формою вірші на основі зібраного дата сету.

Для зручності розглянемо критерії детальніше.

Контекст – у вірші моделюється послідовний розвиток взаємопов’язаних подій.

Рима – останні склади слів у кінці строфи є ідентичними або вимовляються однаково.

Метр – виконується чергування складів для легкості читання віршів.

Іншими словами ми повинні створити модель нейронної мережі, яка буде створювати словник слів, який буде враховувати положення слова в строфі відносно інших сліві і у різних контектсах. Крім цього модель повинна навчитися правил граматики, римування та слідкувати за метрикою.

Розглянемо приклад такої задачі. Нехай ми вже натренували нашу модель на зібраному дата сеті. Ввівши рядок вірша, програма повинна його продовжити, створити справжній вірш. Підбір слів повинен відбуватися послідовно в зворотньому напрямку з кінця, враховуючи основні критерії.

Розглянемо математичну постановку задачі та метод за допомогою якого цю задачу можна розв'язати.

3.2 Математична постановка задачі

Нехай маємо множину віршів *X*, що складається із довільних віршів:

$$X = \{x_1, x_2, ...x_n\}, \quad (3.1)$$

де x_i – конкретний вірш, $i = \overline{1,n}$, n – кількість віршів.

Нехай кожний вірш складається з множини слів *W*:

$$W = \{w_{11}, w_{12}, ...w_{mn}\}, \quad (3.2)$$

де $w_{ij}, i = \overline{1,m}, j = \overline{1,n}$ слово, яке стоїть в *i*-й строфі і *j*-й позиції.

m – кількість строф; n – кількість слів у строфі.

Множина слів створюють вірш x, кожен вірш є компонентом дата сету. Задача генерації віршів методами машинного навчання полягає у тому, аби побудувати модель F, яка на вхід приймає рядок слів $input = \{w_{11}, w_{12}, ...w_{1n}\}$ і генерує продовження у вигляді вірша, дотримуючись всіх основних задач:

Задача підбору рими.

Нехай останнє слово рядка w має вигляд $w_1 - w_2 - \cdots - w_n - w_2 - \cdots - w_n$, а рядка v має вигляд $v_1 - v_2 - \cdots - v_m$, де $w_i\ (i = 1, \dots n)$ та $v_j\ (j = 1, \dots m)$ – склади відповідних слів. Тоді задача римування полягатиме у пошуку таких слів, щоб останні склади співпадали, чи були співзвучні, тобто $w_n = v_m$.

Задача дотримання метру.

Кожному рядку w вірша поставимо у відповідність вектор $(w_1, w_2, \dots, w_n)$, де $w_i\ (i = 1, \dots n)$ – склади слів у рядку. Координати даного вектора набувають значень 0 або 1. 0 – ненаголошений склад, 1 – наголошений.

$w \to \{w_1, w_2, \dots, w_n\}$

Згідно законів поезії, зокрема закону альтерансу, для строфи з чотирьох рядків існують такі можливі схеми римування (**ААбб, ааББ, АббА, аББа, аБаБ, АбАб**), де великими літерами позначаються жіночі рими – коли рядку ставиться у відповідність вектор з передостанньою координатою 1 (з передостаннім наголошеним складом), а маленькими – чоловічі – коли рядку ставиться у відповідність вектор з останньою координатою 1 (з останнім наголошеним складом). Тобто кожній строфі вірша ставиться у відповідність один із елементів цієї цієї множини.

$\mathbf{x} \rightarrow \mathbf{x_i}$, де $\mathbf{x_i} \in$ {ААбб, ааББ, АббА, аББа, аБаБ, АбАб}

У подальшому у вірш об’єднуються строфи з однаковою схемою римування.

Задача дотримання єдиного контексу.

Встановити зв’зок між словами за допомогою векторного представлення:(дерево - фактори, які входить у вектор)

$$w_{ij} = \begin{pmatrix} v_1 \\ \dots \\ v_n \end{pmatrix}, \qquad (3.4)$$

де v_i – координати слова у n-вимірному просторі для встановлення зв’язків між ними. Основні критерії для формування значення слова у контексті зображено на рисунку 3.1.

Рисунок 3.1 – Основні компоненти для встановлення контесту слова

Генератором F зветься визначена на множині віршів X, функція яка отримує на вхід рядок і генерує слова для продовження вірша.

$$F(input) \rightarrow \{next\_word\} \quad (3.5)$$

3.3. Обґрунтування методу розв'язання

Для вирішення цієї задачі ми використовуватимемо методи машинного навчання – нейронні мережі. Планується використати модель нейронної мережі трансформер з механізмом уваги.

3.3.1 Опис прихованої марковської моделі (Hidden Markov Model)

Спочатку коротко розглянемо ланцюги Маркова, які є основою для поточної моделі. Почнемо з кількох «станів» ланцюга, $\{s_1,...,s_k\}$; Наприклад, якщо наш ланцюг представляє слова у рядку вірша: {Someday, you, will, cry}. Властивістю процесу $(X_t)_t$ повинен бути ланцюг Маркова [12]:

$$P(X_t=j|X_1=i_1,...,X_{t-1}=i_{t-1})=P(X_t=j|X_{t-1}=i_{t-1}) \quad (3.6)$$

Тобто, ймовірність опинитися у стані j залежить лише від попереднього стану, а не від того, що сталося раніше. Ланцюги Маркова часто описуються графом з ймовірністю переходу, тобто ймовірністю переходу в стан j із стану i, що позначаються $p_{i,j}$. Розглянемо наступний приклад:

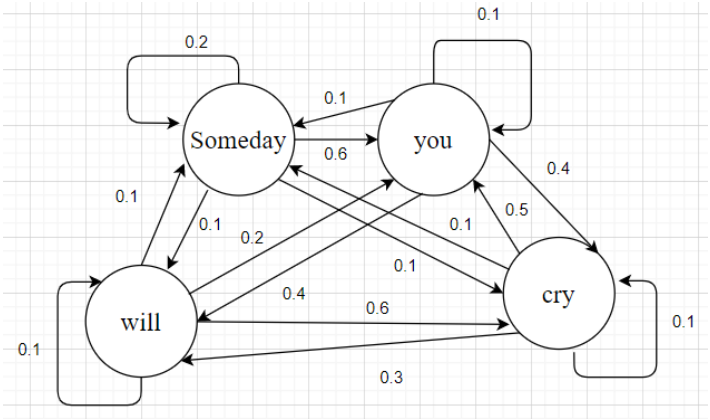


Рисунок 3.2 – Марковський ланцюг

Ланцюг має чотири стани.

Наприклад, ймовірність переходу між you та cry становить 0,4, тобто – якщо поточне слово - you, тобто з ймовірністю 0,4 наступним словом буде will або cry. Ймовірності переходу можна записати у вигляді матриці:

$$P = \begin{pmatrix} 0.2 & 0.6 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.4 & 0.4 \\ 0.1 & 0.2 & 0.1 & 0.6 \\ 0.1 & 0.5 & 0.3 & 0.1 \end{pmatrix} \quad (3.7)$$

Зауважте, що сума кожного ряду дорівнює 1. Така матриця називається стохастичною матрицею. *(i, j)* визначається як $p_{i,j}$ - ймовірність переходу між *i* і *j*. Якщо ми візьмемо потужність матриці P^k, то *(i, j)* представляє ймовірність переходу зі стану *i* в стан *j* на *k* кроках.

У більшості випадках на вхід надходить вектор початкових ймовірностей $q = (q_1, \ldots, q_k)$, що знаходяться в кожному стані в момент $t = 0$. Таким чином, ймовірність перебувати у стані *i* в момент часу *t* буде дорівнювати *i*-му запису вектора $P^k q$.

Наприклад, якщо на поточному кроці ймовірності слів: Someday, you, will, cry становлять 0.1,0.4,0.2,0.3, то ймовірність появи слова you через 7 кроків обчислюється так:

$$P = \begin{pmatrix} 0.2 & 0.6 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.4 & 0.4 \\ 0.1 & 0.2 & 0.1 & 0.6 \\ 0.1 & 0.5 & 0.3 & 0.1 \end{pmatrix}^7 \cdot (0.1,0.4,0.2,0.3)^t \quad (3.8)$$

Цей підхід не задовільняє задачу, яка пов’язана з контекстом, бо даний алгоритм враховує тільки попередній стан. Це робить неможливим збереження єдиної сюжетної лінії.

Ланцюги Маркова корисні, коли нам потрібно обчислити ймовірність послідовності спостережуваних подій. Однак у багатьох випадках події, які нас цікавлять, приховані: ми їх не спостерігаємо безпосередньо. Наприклад,

ми зазвичай не задумуємося про частини мови в тексті. Ми можемо зробити аналіз речення, якщо будемо думати про це цілеспрямовано. Цей процес є прихованими при створенні текстових послідовностей, оскільки цей процес не спостерігається явно. Прихована Марковська модель (HMM) дозволяє нам говорити як про спостережувані події (слова, які ми бачимо на вході), так і приховані події (частини мови).

Hidden Markov Model (HMM) заснований на розширенні ланцюга Маркова. Ланцюги Маркова - це модель, яка описує ймовірності послідовностей випадкових змінних, станів, кожна з яких може приймати значення з деякого набору. Ці набори можуть бути словами, символами чи іншою сутністю. Ланцюги Маркова припускають, якщо ми хочемо передбачити майбутнє в послідовності, все важливе - це поточний стан. Минулі стані перед поточним станом не впливають на майбутні - тільки поточний стан. Це ніби передбачити завтрашню погоду, ви можете знати сьогоднішню погоду, але вам не потрібно дивитися на вчорашню погоду [13].

Іншими словами Прихована Марковська модель (HMM) - це статистична модель, яка може бути використана для опису еволюції спостережуваних подій, які залежать від внутрішніх факторів, які безпосередньо не спостерігаються. HMM складається з двох стохастичних процесів, а саме: невидимого процесу прихованих станів і видимого процесу помітних символів. Приховані стани утворюють ланцюг Маркова, і розподіл ймовірності спостережуваного символу/слова залежить від основного стану. З цієї причини HMM також називають подвійно вбудованим стохастичним процесом. У задачі генерування віршів нам цікаво передбачити наступне слово, аналізуючи попередні. З цією метою генератор намагається знайти послідовність фонем (станів) і встановити залежність між ними, які породили останнє слово.

Сформулюємо математичну постановку НММ. Позначимо спостережувану послідовність слів як $x = x_1 x_2 \ldots x_L$, а послідовність базового стану як $y = y_1 y_2 \ldots y_L$, де y_n. - базовий стан *n*-го спостереження x_n. Кожне слово x_n приймає кінцеву кількість можливих значень із набору спостережень $O = \{o1, o2, ..., oN\}$ і кожен стан y_n приймає одне із значень із набору станів $S = \{1,2,...,M\}$, де N і M позначають кількість різних спостережень і кількість різних станів у моделі відповідно. Ми припускаємо, що послідовність прихованих станів є однорідною за часом Марковським ланцюгом. Це означає, що ймовірність введення стану *j* у наступний момент часу залежить лише від поточного стану *i*, і що ця ймовірність не змінюється з часом. Тому маємо наступне рівняння:

$$\mathrm{P}\{y_{n+1} = \mathrm{j} \mid y_n = i, y_{n-1} = i_{n-1}, ..., y_1 = i_1\} = \mathrm{P}\{y_{n+1} = \mathrm{j} \mid y_n = \mathrm{i}\ \} = t(i, j) \qquad (3.9)$$

для всіх станів $i, j \in S$ і для всіх $n \geq 1$. Фіксовану ймовірність здійснення переходу зі стану *i* в стан *j* називають ймовірністю переходу, і позначимо її як *t (i, j)*. Для початкового стану y_1 позначимо ймовірність початкового стану як $\pi(i) = P\{y1 = i\}$ для всіх $i \in S$. Ймовірність того, що *n*-е спостереження буде $x_n = x$, залежить лише від базового стану y_n, звідси

$$\mathrm{P}\{x_n = x \mid y_n = i, y_{n-1} = x_{n-1}, ...\} = \mathrm{P}\{x_n = x \mid y_n = \mathrm{i}\ \} = e(x, i) \qquad (3.10)$$

для всіх можливих спостережень $x \in O$, всі стани $i \in S$, і всі $n \geq 1$. Це називається emission ймовірністю *x* у стані *i*, і позначимо як *e (x | i)*. Три міри ймовірності *t(i,j)*, *π(i)* і *e(x|i)* повністю характеризують НММ. Для зручності позначимо набір цих параметрів як Θ.

Виходячи з цих параметрів, ми можемо тепер обчислити ймовірність того, що НММ буде генерувати послідовність спостереження $x = x_1 x_2 \ldots x_L$ з послідовністю базового стану $y = y_1 y_2 \ldots y_L$. Ця спільна ймовірність $P = \{x, y \mid \Theta\}$ можна обчислити по формулі 3.11.

$$P = \{x, y \mid \Theta\} = P\{x \mid y, \Theta\} P\{y \mid \Theta\} \quad (3.11)$$

Розпишемо детальніше елементи попередньої формули.

$$P = \{x, y \mid \Theta\} = e(x_1 \mid y_1) e(x_2 \mid y_2) e(x_3 \mid y_3) ... e(x_L \mid y_L), \quad (3.12)$$

$$P = \{y \mid \Theta\} = \pi(y_1) t(y_1 \mid y_2) t(y_2 \mid y_3) ... t(y_{L-1} \mid y_L). \quad (3.13)$$

Як ми бачимо, обчислення ймовірності спостереження є простим, коли ми знаємо послідовність основного стану [14].

Розглянемо використання даної моделі на прикладі задачі римування:

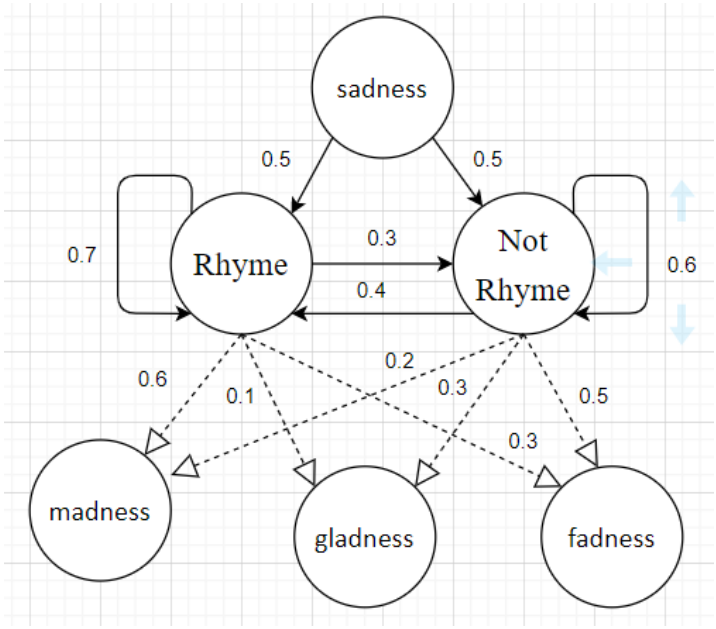


Рисунок 3.3 – Приклад HMM для вирішення задачі римування

Введемо параметри для нашої моделі.

T = довжина послідовності спостереження.

N = кількість станів у моделі.

M = кількість символів спостереження.

Q = $\{ y = q_0 q_1 ... q_{N-1} \}$ = окремі стани марківського процесу.

V = $\{0,1, ..., M-1\}$ = сукупність можливих спостережень.

A = ймовірність переходу стану.

B = матриця ймовірності спостереження.

π = початковий стан розподілу.

O = ($o_0 o_1 \dots o_{T-1}$) = послідовність спостереження.

Нехай T = немає спостережень, N = 2, M = 3, Q = {"Rhyme", "Not Rhyme"}, V = {"madness", "gladness", "fadness"}. Ймовірності переходу стану - це стрілки, що вказують на кожен прихований стан. Матриця ймовірностей спостереження - це пунктирні стрілки, що вказують на кожне спостереження з кожного прихованого стану. Матриця є стохастичними рядками, тобто сума по рядку дорівнює 1.

Матриця пояснює, якою є ймовірність переходу в один стан до іншого або переходу від одного стану до спостереження. Повна модель з відомими ймовірностями переходу стану, матриця ймовірностей спостереження та початковий розподіл стану позначені як:

$$\lambda = (A, B, \pi) \tag{3.14}$$

У вищенаведеному випадку emission дискретні {"madness", "gladness", "fadness"}.

Для пошуку прихованих станів використовується алгоритм Вітербі.

У деяких випадках ми маємо ряд спостережень і хочемо знайти найбільш ймовірні відповідні приховані стани.

Рішення перебором зайняло б експоненціальний час. Більш ефективний підхід називається алгоритмом Вітербі [14], його основна ідея полягає в наступному: нам дається послідовність спостережень $o_1, \dots, o_t$. Для кожного стану i і $t = 1, \dots, T$ визначаємо

$$\eta_t(i) = \max P\{x_1 = i_1, \dots, x_{t-1} = i_{t-1}, x_t = i, o_1, \dots, o_t\} \tag{3.15}$$

Тобто максимальна ймовірність послідовості, яка закінчується в момент часу t у стані i, враховуючи спостереження. Основне зауваження тут полягає в тому, що за властивістю Маркова, якщо найбільш вірогідний шлях, який закінчується на i в момент часу t, дорівнює деякому i^* в момент $t-1$, то

i^* - значення останнього стану найбільш вірогідної послідовності, яка закінчується в часі t−1. Це дає нам наступну рекурсію:

$$\eta_t(i) = \max_i p_{i,j} a_j(o_t) \eta_{t-1}(i) \quad (3.16)$$

$\alpha_j(o_t)$ - ймовірність отримати o_t, коли прихований стан Маркова дорівнює j.

Навчання та алгоритм Баума-Велша

Аналогічний підхід до описаного вище може бути використаний для коригування параметрів моделі HMM під час навчання. У нас є деякий набір даних, і ми хочемо знайти параметри, які будуть найкращими для моделі HMM. Алгоритм Баума-Велша - це ітераційний процес, який знаходить (локальний) максимум ймовірності спостережень $P(O|M)$, де M це модель (з параметрами, які ми хочемо підібрати). Оскільки ми знаємо $P(M|O)$, ми можемо використовувати байєсівський підхід для пошуку $P(M|O)$ та наближення до оптимального значення. [14]

У HMM виникають такі ключові проблеми, які нелегко вирішити.

Проблема 1. З огляду на відому модель, яка ймовірність виникнення послідовності O.

Проблема 2. З огляду на відому модель і послідовність O, яка оптимальна послідовність прихованого стану. Це стане в нагоді, якщо ми хочемо знати, чи слово римується чи ні.

Проблема 3. З огляду на послідовність O та кількість прихованих станів, яка оптимальна модель, яка максимально збільшує ймовірність O [15].

3.3.2 Опис нейронних мереж (Neural Networks)

Іншим популярним методом розв’язання поставлених задач є нейронні мережі. Різні моделі ефективні в різних завданнях моделювання мови. Є дві основні мовні моделі: Word-RNN – модель, де найменшою оперуючою одиницею є слово, є досить ефективною у захопленні семантичного

значення; Char-RNN - модель, де найменшою оперуючою одиницею є символ, потенційно містять більше інформації про морфеми, римування та метр. Моделі на основі слова мають більш високу точність та потребують менших обчислювальних витрат, ніж мовні моделі на основі символів. Це пояснюється тим, що мовна модель RNN на основі символів потрібно значно більше число прихованих шарів, щоб успішно моделювати довгострокові залежності, це означає більш високі обчислювальні витрати. Але мовні моделі на основі слів навчаються швидше та генерують більш узгоджені тексти, але навіть ці генеровані тексти далеко не мають фактичного сенсу. Наприклад, для прогнозування третього слова з початку речення, RNN на основі слова має зробити два передбачення, щоб потрапити туди, тоді як RNN на рівні символів повинен робити прогнози *n* кількість разів, рівну кількості символів перед другим пробілом. Чим більше передбачень має зробити RNN, тим більше модель буде схильною до помилок. Також тренування ускладнюється.

Якщо використовується механізм уваги, то матриця уваги також набагато більша в RNN на основі символів, ніж у RNN на основі слова тому, що в реченні більше символів, ніж слів.

RNN / LSTM

Рекурентні нейронні мережі та моделі довготривалої пам'яті майже однакові за своїми основними властивостями:

- послідовна обробка: рядки повинні бути оброблені слово за словом.
- інформація про попередні слова зберігається через приховані стани: виконується властивість Маркова, кожний стан вважається залежним лише від попереднього стану [16].

Перша властивість є причиною того, що RNN та LSTM не можуть тренуватися паралельно. Для кодування другого слова в реченні потрібні обчислювані раніше приховані стани першого слова, тому потрібно

обчислити їх спочатку. Інформація в RNN та LSTM зберігається за допомогою раніше розрахованим прихованим станам. Справа в тому, що кодування конкретного слова зберігається лише на наступному кроці часу, а це означає, що кодування слова сильно впливає лише на подання наступного слова, його вплив швидко втрачається через кілька кроків. LSTM (а також GruRNN [17]) може трохи збільшити діапазон залежностей, який вони можуть засвоїти, завдяки більш глибокій обробці прихованих станів через збільшенням кількості параметрів для тренування, але проблема пов'язана з рекурсією залишається. Ще одним способом, яким люди пом'якшили цю проблему, є використання двонаправлених моделей, які кодують одне й те саме речення з двох напрямків - від початку до кінця та від кінця до початку, дозволяючи таким чином словам в кінці речення мати сильніший вплив у створенні прихованого представлення, але це лише лазівка, а не реальне рішення для дуже довгих залежностей.

Ядро LSTM випливає з рівняння, вираженого в рівнянні 3.17.

$$c_t = f_t * c_{t-1} + i_t * g_t \tag{3.17}$$

Стан комірки, c_t, служить пам'яттю LSTM і дозволяє RNN відслідковувати довгострокові залежності. Потім комірка LSTM вибирає зберегти частину своєї попередньої інфомації/стану комірки через ворота забуття f_t, проілюстровані в рівнянні 3.18. Ще одна важлива складова - це вхідні ворота i_t, диктує як багато оновленої інформації включено до оновленої комірки стан з попереднього виводу комірок, як проілюстровано у рівнянні 3.19. g_t представляє ворота, що містять використану інформацію для оновлення c_t, в той час як h_t служить вихідним елементом LSTM в момент часу *t*, як це визначено у рівнянні 3.21.

$$f_t = \sigma(W_{if} x_t + b_{if} + W_{hf} h_{t-1} + b_{hf}), \tag{3.18}$$

$$i_t = \sigma(W_{ii} x_t + b_{ii} + W_{hi} h_{t-1} + b_{hi}), \quad (3.19)$$

$$g_t = \tanh(W_{ig} x_t + b_{ig} + W_{hg} h_{t-1} + b_{hg}), \quad (3.20)$$

$$h_t = \sigma(W_{io} x_t + W_{ho} h_{t-1} + b_o) * \tanh(c_{t-1}). \quad (3.21)$$

На відміну від стандартної архітектури RNN, комірка LSTM вибирає кількість інформації, яку потрібно зберегти від попередніх комірок і що додати до поточного стану комірки, c_t. Можливість адекватно вчитися дозволяє коміркам LSTM захоплювати довгострокові залежності набагато ефективніше, ніж стандартні RNN. Хоча мовні моделі LSTM добре працюють, але навчання їх на великих наборах даних дуже трудомісткий через нормалізацію розподілу передбачуваних слів. Цей процес змушує модель розглядати кожне слово в словнику при обчисленні градієнтів. GRU вимагає меншої кількості тензорних операцій для навчання, це прискорює навчальний процес і забезпечує альтернативу LSTM. Застосування цієї методики моделювання може кардинально прискорити наш навчальний процес. Також останніми роками LSTM на основі уваги досягли вражаючих результатів при вирішенні задач з текстовими послідовностями [19].

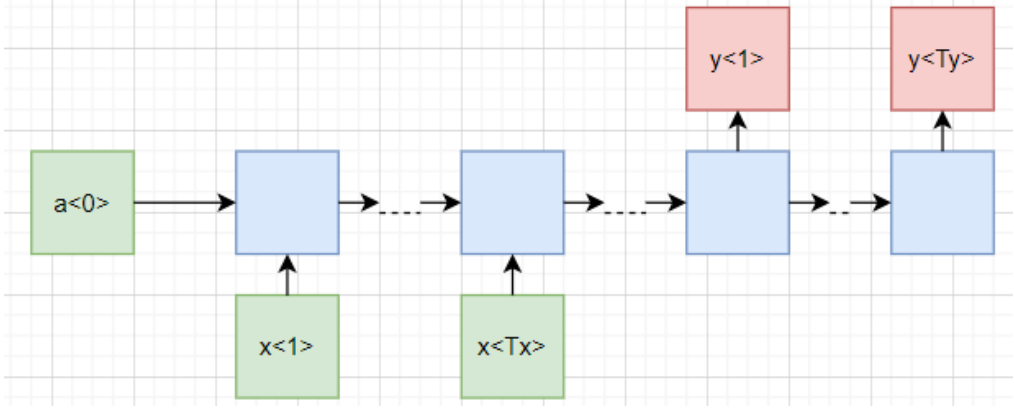


Рисунок 3.4 – Модель рекурентної нейронної мережі *Tx≠Ty* [20]

Для кожного моменту часу *t* функція активації *a<t>* і вихідне значення *y<t>* можна виразити як:

$$a^{<t>} = g_1(W_{aa} a^{<t-1>} + W_{ax} x^{<t>} + b_a) \quad (3.22)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y), \quad (3.23)$$

де $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ коефіцієнти, які тимчасово діляться між комірками, а W_{aa}, W_{aa} - функції активіації.

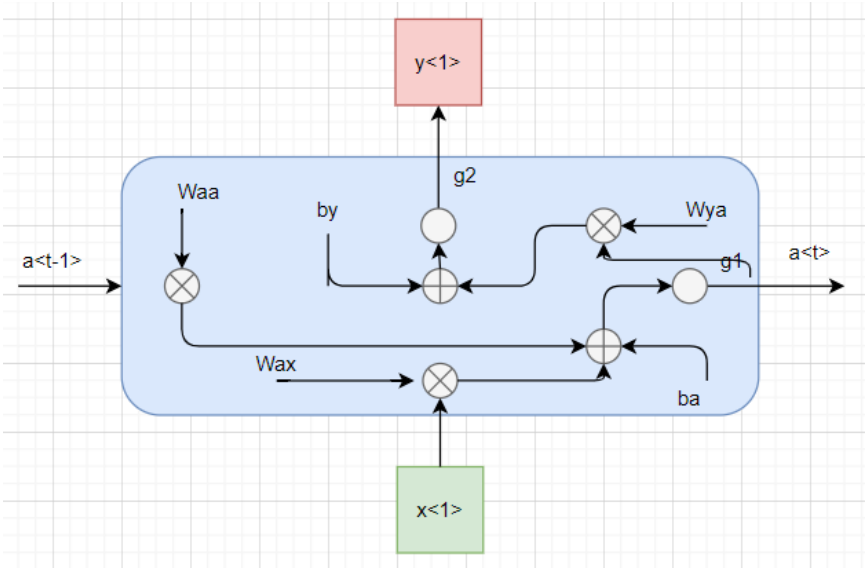


Рисунок 3.5 – Модель комірки рекурентної нейронної мережі

Добре відома проблема - зникнення / вибуху градієнти. Це означає, що модель є упередженою більшістю останніх входів у послідовності, або іншими словами, старі входи практично не впливають на вихід на поточному кроці. В основному LSTM / GRU [18] намагаються вирішити цю проблему, включаючи окрему пам'ять (комірку) та / або додаткові ворота, щоб дізнатися, коли відпустити минулу / поточну інформацію.

CNN

Згорткова нейронна мережа широко використовується в nlp, оскільки вони досить швидкі для навчання та ефективної роботи з короткими текстами. Спосіб вирішення залежностей полягає в застосуванні різних ядер до одного і того ж речення. Чому різні ядра дозволяють дізнатися залежності? Оскільки ядро розміру 2, наприклад, вивчило б зв'язки між парами слів, ядро розміром 3 захоплювало б зв’язки між трійками слів тощо. Очевидна проблема тут полягає в тому, що кількість різних ядер, необхідних

для фіксації залежностей між усіма можливими комбінаціями слів у реченнях, була б величезною і непрактичною через збільшення експоненціально зростаючої кількості комбінацій при збільшенні максимального розміру вхідних пропозицій [18].

Модель трансформера.

Трансформери були створенні для покращення машинного перекладу з метою уникнення рекурсії, щоб дозволити паралельні обчислення (скоротити час навчання), а також зменшити неточність для великих залежностей. Основні характеристики:

- не послідовний: рядок опрацьовуються як цілий, а не слово за словом;
- увага до самого себе: це нещодавно введена 'одиниця', яка використовується для обчислення балів подібності між словами в реченні;
- позиційний ембедінг: ще одне нововведення, запроваджене на зміну рекурсій. Ідея полягає у використанні фіксованих чи вивчених ваг, які кодують інформацію, що стосується конкретного положення лексеми у рядку.

Перша властивість - головна причина, чому трансформери не страждають від проблем, які виникають при довгих залежностях. Трансформери не покладаються на минулі приховані стани, щоб оцінити залежності з попередніми словами, вони обробляють речення в цілому, тому немає ризику втратити (або «забути») минулу інформацію. Крім того, multi-head та positional embeddings надають інформацію про зв’язок між різними словами.

Однією з головних переваг архітектури трансформерів є те, що на кожному кроці ми маємо прямий доступ до всіх інших кроків (самоуваги), що практично не залишає місця для втрати інформації, що стосується передачі повідомлення. На додаток до цього ми можемо одночасно

розглядати як майбутні, так і минулі елементи, що також приносить користь двонаправленим RNN, без необхідності 2-х обчислення. І звичайно, все це відбувається паралельно, що робить і тренування / умовивід набагато швидшими. Застосовувати трансформери на довгих послідовностях буде дорого, порівняно з RNN. Це єдиний недостаток перед RNN [18].

Можна зробити висновок, що трансформери є кращими за всі інші архітектури, оскільки вони повністю уникають рекурсії, обробляють речення в цілому та вивчаючи взаємозв'язки між словами, завдяки механізмам уваги з multi-head та positional embeddings. Тим не менш, слід зазначити, що трансформери також можуть захоплювати лише залежності в межах фіксованого вхідного розміру, використовуваного для їх навчання, тобто якщо я використовую як максимальний розмір строфи 20, модель не зможе зафіксувати залежності між першим словом речення та слова, які зустрічаються понад 20 слів пізніше. Нові трансформери, такі як Transformer-XL, намагаються подолати саме цю проблему шляхом своєрідного повторного введення рекурсії, зберігаючи приховані стани вже закодованих пропозицій, щоб використовувати їх у подальшому кодуванні наступних речень.

3.4 Опис методів розв’язання

Досліджуючи послідовність слів, ми спостерігаємо, що деякі слова в ній тісніше пов'язані між собою, ніж інші (певні комбінації слів частіше зустрічається). Це породжує концепцію самоуваги (self-attention), в якій кожне слово співставляється до інших слів в послідовності. Тобто, увага приділяється моделюванню контексту, встановлюючи ваги словам, які відображують відносини між ними.

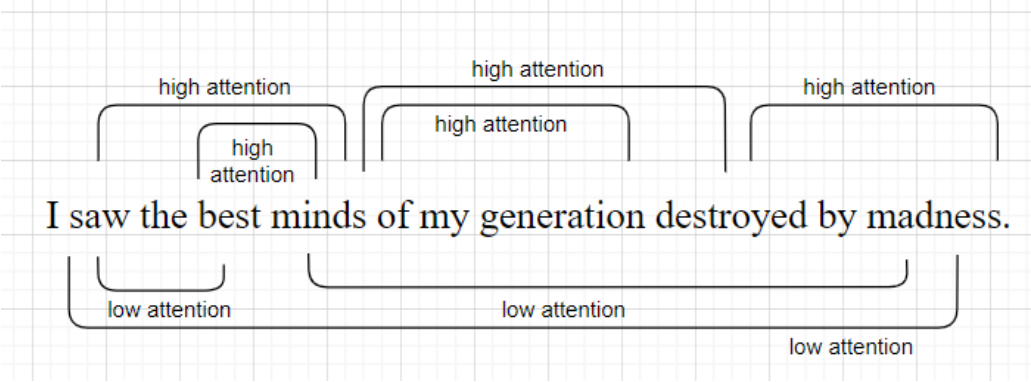


Рисунок 3.6 – Приклад розподілення уваги і рядку вірша

Трансформер - це нейронна мережева архітектура, яка використовує механізм самоуваги. Це змінює більш ранні підходи запропоновані у LSTM або CNN, які використовували увагу між кодером і декодером. На основі цієї архітектури були створені такі мережі, як BERT та GPT-2, які були натреновані на великих дата сетах і мають можливості для додаткової надстройки.

Може виникнути питання: чому мережа трансформер краща за CNN, RNN або LSTM для вирішення поставлених задач. Слова у реченні розташовані одне за одним – послідовно, контекст поточного слова визначається словами, що його оточують. RNN підходять для моделювання такої задачі, але мають проблеми з запам'ятовуванням довгих послідовностей. LSTM - це варіант RNN, який є кращим у цьому плані, але теж має свої недоліки. Архітектури CNN WaveNet, ByteNet і ConvS2S також використовуються для дослідження таких послідовностей. Крім того, RNN та LSTM розглядають лише попередні слова, які були раніше (хоча є двонаправлені LSTM). Самоувага моделює контекст, переглядаючи попередні і наступні слова відносно поточного слова. Наприклад, слово "object" у реченні "he objects me" не є іменником, а вживається як дієслово. Трансформер може зрозуміти це значення у поточному контексті, оскільки він переглядає попередні і наступні слова.

Основний недолік RNN полягає в тому, що завдання не можуть бути паралелізованими. Самоувагу кодера трансформера можна паралелізувати. Це дозволяє зменшити час навчання моделі в рази, взалежності від характеристики обчислювальної машинии. CNN менш послідовна, але складність все ще зростає логарифмічно. Це сповільнить процес навчання на великих дата сетах. Гірше для RNN - складність зростає лінійно. Для трансформера кількість послідовних операцій є постійною.

Модель трансформера складається з кодер-декодер з увагою, що передається від кодера до декодера. І кодер, і декодер складаються з кількох однакових шарів. Кожен шар кодера використовує увагу для представлення контексту. Кожен шар декодера також використовує увагу в двох підшарах. У той час як самоувага кодера використовує лівий і правий контекст, нижній підшар декодера маскує майбутні позиції, прогнозуючи поточну позицію. У кожному шарі ми знаходимо деякі загальні елементи, будуються залишкові з'єднання. Вони додаються та нормалізуються за допомогою з'єднань, що пролягають через підшари самоуваги. Можемо побачити, що тут не створюється рекурентна мережа, лише повністю з'єднана мережа з прямим зв'язком. На вході вхідні та вихідні послідовності представлені як векторне представлення слів. Вони покращуються за допомогою позиційних кодувань. На виході розташований лінійний шар з softmax для проведення згладжування результатів. Кодер трансформера може працювати паралельно над послідовністю введення, але декодер є авторегресивним. На кожен вихід впливають попередні слова/символи виводу. Слова/символи виводу генеруються по одному.

Для обчислення самооцінки в трансформері використовується наступна схема. Для кожного слова створюються три вектори: запит (query), ключ (key) та значення (value) [21]. Відповідні вагові матриці W обчислюються під час тренувань. Давайте змоделюємо, що ми обчислюємо увагу для певного

слова. Обчислюється скалярний добуток вектора запиту з вектором-ключом кожного слова. Увага скалярного добутку масштабується як $1/\sqrt{d_k}$ для компенсації великих значень скалярного добутку. Вектори значень порівнюються з вагами скалярного добутку і потім підсумовуються. Для кращих результатів використовується багатоголова увага (multi-head attention) [21]. Кожна голова вчиться різному розподілу уваги, подібно до того, як фільтри в CNN. Наприклад, якщо розмір моделі становить 512, замість великого одного шару уваги ми використовуємо 8 паралельних шарів уваги, кожен з яких працює в 64 вимірах. Вихід з шарів з'єднаний, щоб отримати остаточну увагу. Трансформер використовує самоувагу в кодері та декодері, але також передає увагу від кодера до декодера, як це прийнято в традиційних моделях послідовность-до-послідовность (sequence-to-sequence models).

Розберемо як мережа трансформер фіксує відносне положення слів у послідовностях. У мережі самоувага ігнорує положення лексем у межах послідовності. Щоб подолати це обмеження, трансформери явно додають позиційні кодування. Вони додаються до вхідних чи вихідних векторів до того, як сума цих значень буде передана в перший рівень уваги. Позиційні кодування можуть бути вивчені або зафіксовані. В останньому випадку використовуються синусоїдальні та косинусні функції для парних і непарних позицій відповідно. Вони також використовують різні частоти для різних позицій, щоб полегшити для моделі вивчення позицій. Також дивляться на відстань між лексемами в послідовності, тобто відносне розташування. Вони показали, що це призводить до кращих результатів із роботою з текстом на 7% на кожному кроці. Розглянемо роботу мережі детальніше.

Увага (attention) - це концепція, яка допомогла покращити продуктивність нейронних мереж у роботі з текстом. «Трансформер» -

модель, яка використовує увагу, щоб підвищити швидкість тренування моделі. Найбільша користь полягає в тому, що ця модель піддається паралелізації. Трансформер був запропонований у статті «Увага - це все, що вам потрібно». Реалізація його TensorFlow доступна як частина пакету Tensor2Tensor.

Для початку розглянемо модель як чорну скриньку. У моделі для генерування вірша на вхід подається послідовність рядків, а на виході отримуємо слово - продовження послідовності.

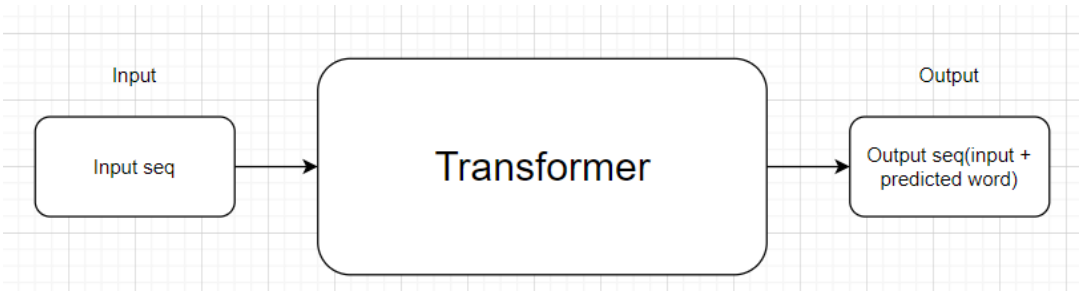


Рисунок 3.7 – Приклад моделі трансформера

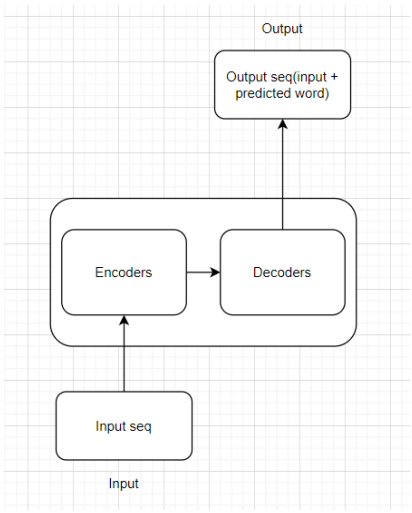


Рисунок 3.8 – Будова трансформера

Компонент кодування - це стек кодерів (число визначається гіпермараметром, який визначається експериментально). Компонент декодування - це стек декодерів такої ж кількості як і кодерів.

Всі кодери однакові за структурою (проте вони не мають загальних/спільних ваг). Кожен з них розбивається на два підшари. Вхідні

слова, які надходять в кодер спочатку проходять через шар самоуваги (self-attention) - шар, який допомагає кодеру переглядати інші слова у вхідному реченні, оскільки він кодує певне слово. Вихідні результати шару самоуваги подаються в нейронну мережу, направлену вперед (feed forward). Для кожної позиції застосовується така сама мережа передачі даних. Декодер має обидва ці шари, але між ними є шар уваги (attention), який допомагає декодеру зосередитись на відповідних частинах вхідного речення (подібно до того, що привертає увагу в моделях seq2seq).

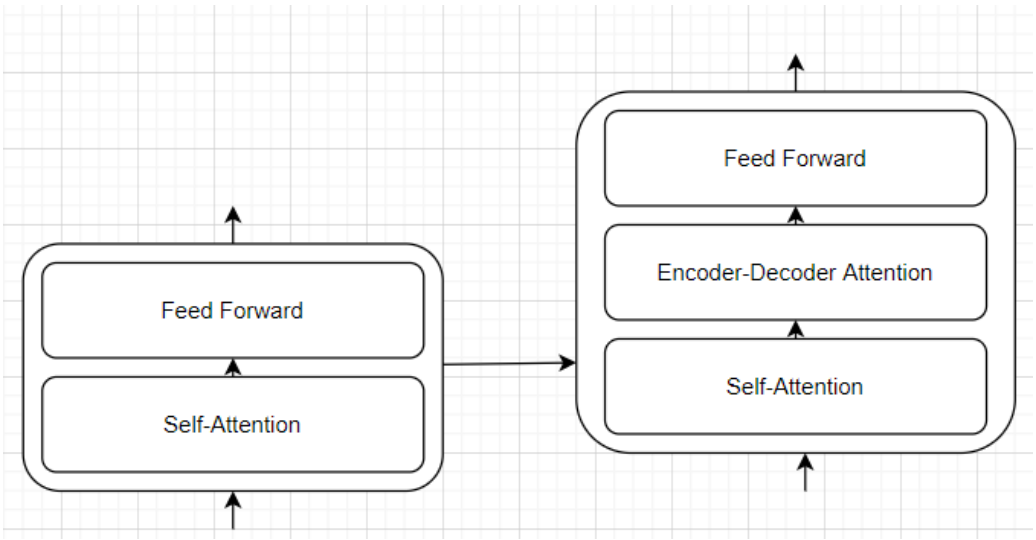


Рисунок 3.9 – Структура кодера та декодера

Перед тим як передати слова в кодер відбувається перетворення слів у вектор. Вони отримують список векторів розмірністю 512. Розмірність визначається як гіперпараметр, який ми можемо встановити - в основному це буде довжина найдовшого вірша в нашому навчальному наборі даних. Після перетворення слів у вектори, то кожен з них проходить через кожен з двох шарів кодера. Ми можемо спостерігати ключову властивість Трансформера, яка полягає в тому, що слово в кожній позиції протікає своїм шляхом у кодері. Між цими шляхами існують залежності в шарі самооцінювання. Тому різні шляхи можуть виконуватися паралельно під час проходження через feed-forward layer.

Можна встановити просту схему: кодер отримує список векторів у якості вхідних даних. Він обробляє цей список, передаючи ці вектори в шар "самоуваги", потім в нейронну мережу (feed forward neural network), а потім надсилає вихідні дані наступному кодеру [22].

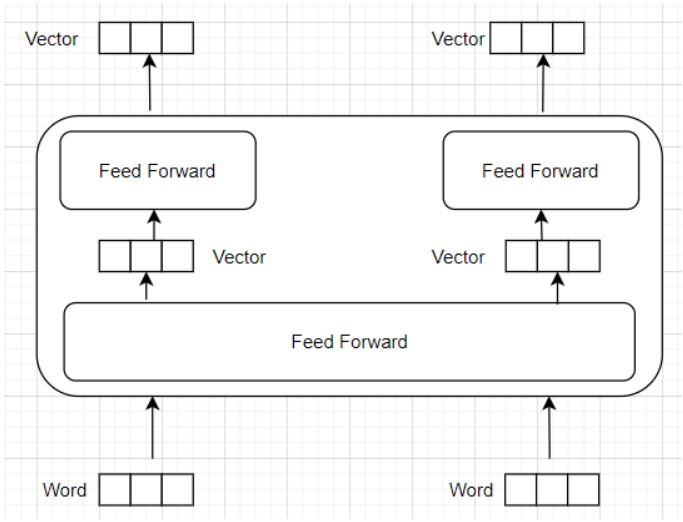


Рисунок 3.10 – Детальна будова кодера

Розглянемо механізм самоуваги на практиці.

Перший крок в обчисленні уваги - створити три вектори з кожного вхідного вектора кодера (у цьому випадку векторизація кожного слова). Отже, для кожного слова ми створюємо вектор запиту, вектор-ключ та значення. Ці вектори створюються шляхом множення векторів на три матриці, які ми отримали під час навчального процесу. Нові вектори мають менші розміри, ніж вектор вбудовування. Їх розмірність становить 64, тоді як вхідні та вихідні вектори мають розмірність 512. Вектори "запит", "ключ" та "значення" корисні для обчислення уваги.

Другий крок у підрахунку самоуваги - це підрахунок оцінки. Потрібно оцінити кожне слово вхідного рядка відносно вибраного слова. Оцінка визначає, скільки уваги потрібно поставити на інші частини вхідного речення, коли ми кодуємо слово на певній позиції.

Оцінка обчислюється як скалярний добуток вектора-запиту з ключ-вектором відповідного слова, яке ми оцінюємо. Отже, якщо ми обчислюємо увагу до слова в позиції №1, першою оцінкою буде скалярний добуток *q1* і *k1*. Друга оцінка - це скалярний добуток *q1* і *k2*.

Третій і четвертий кроки - розділити бали на 8 (квадратний корінь розмірності ключових векторів, використовуваних у роботі - 64. Це призводить до отримання більш стабільних градієнтів. Тут можуть бути й інші можливі значення, але це за замовчуванням), а потім передати результат через операцію softmax. *Softmax* нормалізує оцінки, тому всі вони позитивні і підсумуються до 1. Цей показник softmax визначає, скільки разів кожне слово буде зустрічатися в цій позиції. Очевидно, що слово на цій позиції матиме найвищий показник softmax, але іноді корисно аналізувати інше слово, що відповідає поточному слову.

П’ятий крок - множення кожного вектору-значення на показник softmax. Інтуїція тут полягає в тому, щоб зберегти недоторканими значення слів, на яких ми хочемо зосередити увагу, і заглушити невідповідні слова (множивши їх, наприклад, на крихітні числа, наприклад, 0,001).

Шостий крок - підсумовувати зважені вектори. Це створює вихід шару самоуваги в цьому положенні (для першого слова).

Це завершує розрахунок самооцінки. Результуючий вектор - це вектор, який ми можемо передати в feed-forward мережу. У реальній реалізації, однак, цей розрахунок проводиться в матричній формі для більш швидкої обробки.

Слово	q вектор	k вектор	v вектор	Оцінка	Оцінка/8	SoftMax	Softmax*v	Сума
Слово 1	*q1*	*k1*	*v1*	*q1*k1*	*q1*k1/8*	*x11*	*x11*v1*	Z1
Слово i	*qi*	*ki*	*vi*	*qi*ki*	*qi*ki/8*	*x12*	*x12*v2*	Zi
Слово n	*qn*	*kn*	*vn*	*qi*ki*	*qi*ki/8*	*x13*	*x13*v3*	Zn

Рисунок 3.11 – Обчислення уваги

Механізму уваги може бути описане наступним рівнянням:

$$Attention(Q,K,V) = soft\max(\frac{QK^T}{\sqrt{d_k}})V \quad (3.24)$$

Q - матриця, яка містить запит (векторне представлення одного слова в послідовності), K - всі ключі (векторні подання всіх слів у послідовності) і V - значення, які знову є векторними поданнями всіх слова в послідовності. Для кодера і декодера, багатоголових модулів уваги, V складається з тієї ж послідовності слів, що й Q. Однак для модуля уваги, який враховує кодер і послідовності декодера, V відрізняється від послідовності, представленої Q.

Щоб трохи спростити це, ми могли б сказати, що значення V перемножуються та підсумовуються з деякими вагами a, де наші ваги визначаються:

$$a = soft\max(\frac{QK^T}{\sqrt{d_k}}) \quad (3.25)$$

Це означає, що ваги a визначаються тим, як на кожне слово послідовності (Q) впливають усі інші слова в послідовності (K). Крім того, функція *SoftMax* застосовується до ваг a для розподілу між 0 і 1. Ці ваги потім застосовуються до всіх слів у послідовності, що вводяться в V (ті ж вектори, що і Q для кодера і декодера, але різні для модуль, який має входи кодера і декодера).

Механізм уваги повторюється багаторазово за допомогою лінійних проекцій *Q, K і V*. Це дозволяє системі вчитися з різних представлень *Q, K* і *V*, що є вигідним для моделі. Ці лінійні представлення виконуються шляхом множення *Q, K* і *V* на вагові матриці *W*, які визначаються під час тренування.

Матриці *Q, K і V* відрізняються для кожній позиції модулів уваги в структурі, залежно від того, чи перебувають вони в кодері, декодері або між кодером і декодером. Причина полягає в тому, що ми хочемо ітерувати всю введену послідовність кодера, або частину послідовності декодера. Багатоголовий модуль уваги, який з'єднує кодер і декодер, переконається, що вхідна послідовність кодера враховується разом із вхідною послідовністю декодера до заданої позиції.

Після цього визначається невелика мережа передачі даних, яка має однакові параметри для кожної позиції, які можна описати як окреме, однакове лінійне перетворення кожного елемента із заданої послідовності[23].

Багатоголова увага дозволяє моделі одночасно опрацьовувати інформацію різного представлення, підпростори в різних положеннях. З однією головою уваги процес гальмується.

$$MultiHead(Q,K,V) = Concat(head_1,...,head_h)W^O, \quad (3.26)$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V), \quad (3.27)$$

де проекції - це матриці параметрів $W_i^Q \in R^{d\,\mathrm{mod}\,el * dk}, W_i^K \in R^{d\,\mathrm{mod}\,el * dk}, W_i^V \in R^{d\,\mathrm{mod}\,el * dV}$ і $W^O \in R^{hdvd\,\mathrm{mod}\,el}$.

Крім підшарів уваги, кожен з шарів у кодері та декодері містить мережу, яка складається з двох лінійних перетворень з активацією ReLU між ними:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (3.28)$$

Розглянемо роботу декодера. Кодер починає обробляти введену послідовность. Потім результат верхнього кодера перетворюється на набір векторів уваги *K* і *V*. Вони повинні використовуватися кожним декодером у шарі, що допомагає декодеру зосередитись на відповідних місцях в послідовності введення. Цей процес повторюється, поки не буде досягнутий спеціальний символ, що декодер трансформера не завершив свій вихід. Вихідний результат на кожному кроці подається на нижній декодер і декодери перетворюють свої результати декодування так, як це робили кодери. І так само, як ми робили з вхідними даними кодера, ми векторизуємо слова та додаємо позиційне кодування до цих входів декодера, щоб вказувати положення кожного слова [24].

Шари самоуваги в декодері працюють дещо іншим чином, ніж у кодері. У декодері в шарі самоуваги дозволено ітерувати лише попередні позиції у вихідній послідовності. Це робиться шляхом маскування майбутніх позицій (встановлення їх на -inf) перед викликом функції softmax при розрахунку самоуваги. Шар «Увага енкодера-декодера» працює так само, як шар багатоголової самоуваги, за винятком того, що він створює свою матрицю запитів із шару під ним та приймає матрицю ключів та значень із виходу стека кодера. Стек декодера виводить вектор, який з складається з чисел з плавуючою комою. Розглянемо як ці числа перетворюються в слова. Це завдання останнього лінійного шару, за яким слідує шар Softmax. Лінійний шар являє собою просту нейронну мережу, яка проектує вектор, який був створений стеком декодерів у набагато більший вектор, який називається вектором logits.

Припустимо, що наша модель знає *N* унікальних англійських слів, які вона дізналися з навчального набору даних. Це зробило б векторний логіт шириною N комірок - кожна клітинка відповідає оцінці унікального слова. Саме так ми інтерпретуємо вихід моделі з наступним лінійним шаром. Тоді

softmax шар перетворює ці оцінки у ймовірності (всі позитивні, всі складають до 1,0). Вибирається клітинка з найбільшою ймовірністю, а слово, пов'язане з нею, виробляється як результат для поточного кроку.

Висновок до розділу

Під час написання даного розділу дипломного проекту було наведено змістовну та математичну постановки задачі, визначено основні терміни, які будуть зустрічатися при описі поставлених задач.

В змістовній постановці задачі був наведений приклад, який допоможе детальніше ознайомитися із задачами та можливими методами машинного навчання для їх вирішення.

В математичній постановці задач були сформульовані задачі та описані їх основні атрибути. Визначено математичний запис кожної із задач і глобальної – загальної задачі відповідно.

Також, було наведено опис методів розв’язання задачі, наведено характеристику кожного із методів. Встановлено переваги та недоліки кожного із підходів.

Крім цього було наведено математичне обґрунтування кожного із підходів. Встановлено і доведено, правильність використання обраного методу машиного навчання для вирішення поставлених задач.

Задача, що була представлена у змістовній постановці була розв’язана одним із методів машинного навчання, який стає популярним із кожним роком у роботі з природньою мовою.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для реалізації програмного продукту було проаналізовано різні мови програмування, щоб визначити яка з них найкраще підійде для вирішення поставлених задач та досягнення мети. Після вивчення документацій різних мов і інструментів, була вибрана мова програмування Python для розробки програмного продукту.

Python – високорівнева мова програмування, яка є досить популярною для вирішення задач, пов’язаних з машинним навчанням. Python є інтерпретованою мовою, оскільки програми, написані на Python, виконуються інтерпретатором. Існує два способи використання інтерпретатора: інтерактивний режим і режим скрипту. У інтерактивному режимі розробник взаємодіє з кодом програми Python – пише код програми, а інтерпретатор відображає результат роботи виконаного коду. Крім того, розробник може зберігати код у файлі у вигляді скрипту та підключати його у програмі при необідності, використовувати інтерпретатор.

Також Python має можливість використовувати безліч бібліотек, які спрощують процеч написання коду розробнику, надаючи йому можливість використовувати вже готові реалізації деяких рішень. Це суттєво полегшує процес розробки програмного продукту.

Мова програмування Python є дуже зручною для вирішення задач машинного навчання, бо містить багато бібліотек і фреймворків, що спрощують механізми проведення складних розрахунків, необхідних для побудови моделей машинного навчання, а також забезпечують зручний інструментарій для вирішення поставлених задач.

У розділі 3.4 дипломного проекту «Обґрунтування методу розв’язання» наводився один із підходів розв’язання поставлених задач, який необхідно

використовувати для побудови моделей машинного навчання. Можна відзначити, що цей метод є досить новим, але вже є деякі бібліотеки, які створюють процес створення моделей та реалізації відповідної математики, які підтримують мову програмування Python.

Розглянемо, які бібліотеки та фреймворки були використані під час розробки програмного продукту та наведемо їх коротку характеристику.

Умовно ці бібліотеки можна поділити на 3 типи: бібліотеки, які використовуються для попередньої бробки вхідних даних та створення датасету, створення моделей машинного навчання – нейронних мереж та візуалізація результатів.

Детальну характеристику цих бібліотек наведено у таблиці 4.1.

Таблиця 4.1 – Бібліотеки та фреймворки, що були використані під час розробки програмного продукту

Назва бібліотеки	**Опис**
copy	Модуль для поверхневого та глибокого копіювання об’єктів.
streamlit	Фреймворк із відкритим кодом - це найпростіший спосіб для інженерів машинного навчання створити красиві, якісні програми лише за кілька годин.
hugging face	Фреймворк для роботи з готовими моделями нейронних мереж, які вже є натреновані на великих обсягах даних.
numpy	Бібліотека, яка дає можливість працювати із числовими структурами даних.
tensorflow	Відкрита платформа з відкритим кодом для машинного навчання. Має вичерпну гнучку екосистему інструментів, бібліотек та ресурсів громади, що дозволяє дослідникам підштовхувати найсучасніші в ML та розробники легко створювати та розгортати програми, що працюють на ML..
logging	Модуль для інтеграції логів у програмі. Є корисним для дослідження роботи програми.
psycopg2	Найпопулярніший адаптер бази даних PostgreSQL для мови програмування Python.

Окрім мови програмування Python використовувалися й інші засоби розробки:

- фреймворк streamlit для візуалізації результатів;
- платформа tensorflow для побудови моделі;
- фреймворк hugging face для імпорту ваг натренованих моделей.

Окрім цього для реалізації програмного продукту використовувалася система управління базами даних PostgreSQL.

PostgreSQL - це потужна об'єктно-реляційна база даних із відкритим кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші дані. Розробка PostgreSQL починається в 1986 року в рамках проекту POSTGRES в Каліфорнійському університеті в Берклі і має більш ніж 30 років активного розвитку на основній платформі.

PostgreSQL заслужив потужну репутацію за свою перевірену архітектуру, надійність, цілісність даних, надійний набір функцій, розширюваність та відкритий код, щоб пропонувати і розроблювати ефективні та інноваційні рішення. PostgreSQL працює на всіх основних операційних системах, сумісний з ACID з 2001 року і має потужні додатки, такі як популярний розширювач геопросторових баз даних PostGIS. Не дивно, що PostgreSQL став реляційною базою даних з відкритим кодом для багатьох людей і організацій.

Початок роботи з PostgreSQL ніколи не був простішим - виберіть проект, який ви хочете створити, і дозвольте PostgreSQL безпечно і надійно зберігати ваші дані. PostgreSQL оснащений багатьма функціями, спрямованими на допомогу розробникам у створенні програм, адміністраторам для захисту цілісності даних та побудови середовищ, стійких до відмов, та допомагають керувати вашими [25].

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Даний програмний продукт представлений у вигляді застосування і складається із серверної та клієнтської частин.

Для коректної роботи серверної частини веб застосування необхідно використовувати сервер що може бути представлений локальною обчислювальною машиною. Сервер повинен мати такі конфігурації:

- процесор з тактовою частотою не нижче 2 ГГц;
- достатній об’єм оперативної пам’яті (не менше 4 ГБ);
- жорсткий диск (не менше 40 ГБ);
- операційна система Windows 10;
- база даних PostgreSQL 10.4 і вище;
- підключення до мережі Інтернет;
- Python 3.7;
- Tensorflow 2.0 і вище;
- Stremlit.

Для коректної роботи клієнтської частини веб застосування необхідно:

- підключення до мережі Інтернет;
- заготівки для генерування віршів.

Перша версія програмного забезпечення розробляється для випадку, коли серверною частиною виступає локальна обчислювальна машина, на якій запускається програмний продукт. Тому для коректної роботи клієнтської частини застосування, локальна обчислювальна машина повинна мати такі характеристики:

- процесор з тактовою частотою не нижче 2 ГГц;
- достатній об’єм оперативної пам’яті (не менше 4 ГБ);
- жорсткий диск (не менше 40 ГБ);

- операційна система Windows 10;
- база даних PostgreSQL 10.4 і вище;
- підключення до мережі Інтернет;
- Python 3.7;
- Tensorflow 2.0 і вище;
- Stremlit.

Система повинна адекватно реагувати на помилки серверної частини застосування та видавати відповідні повідомленні користувачеві, а також записувати їх у логи для того, щоб програміст міг дослідити і зрозуміти, де виникла критична ситуація. Це дозволить зменшити ймовірність неправильної роботи системи.

Для коректної роботи системи необхідно забезпечити безперебійне живлення на серверній частині веб-застосування та підключення до мережі Інтернет зі сторони Користувачів системи.

4.3 Архітектура програмного забезпечення

Для даного програмного продукту була обрана клієнт-серверна архітектура. Ця архітектура є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними.

Розглянемо схему структурну архітектури програмного забезпечення. Архітектура програмного забезпечення складається із трьох рівнів: рівень представлення, рівень бізнес логіки, рівень даних.

На рівні представлення знаходиться такий компонент: користувацький інтерфейс, реалізований за допомогою фреймворку streamlit.

Рівень бізнес логіки складається із таких компонентів: Front-end – програмний інтерфейс, з яким взаємодіє користувач, Back-end: модель нейронної мережі, яка оброблює дані і генерує результат (Python 3.7).

Функціонал системи представлений фреймворками та бібліотеками мови програмування Python версії 3.7, а саме:

- numpy (бібліотека для роботи із дата сетом);
- psycopg2 (бібліотека для роботи із сервером баз даних);
- tensorflow (платформа для роботи з моделями машинного навчання);
- streamlit (бібліотека для візуалізації результатів).

На рівні даних реалізовані функції за допомогою яких можна оброблювати інформацію із серверу баз даних.

Головним керуючим компонентом даного програмного забезпечення є сервер. Саме він забезпечує коректну комунікацію між клієнтом і моделею. На рисунку 4.1 зображена модель застосунку.

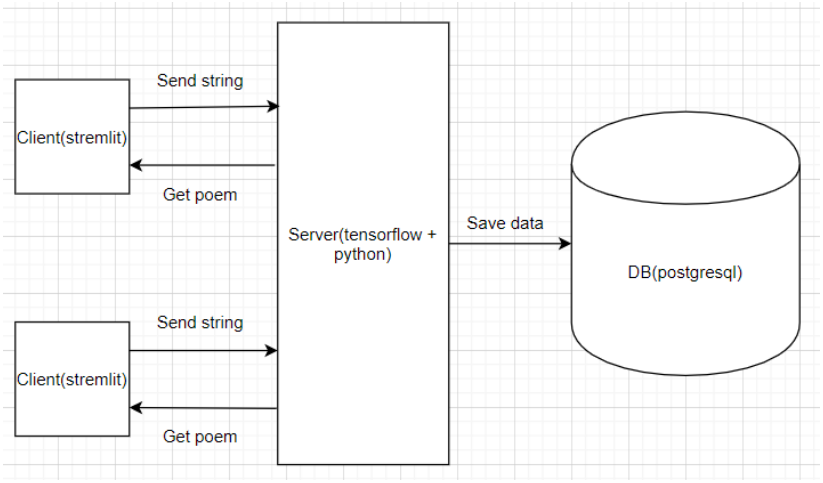


Рисунок 4.1 – Модель застосунку

4.3.1 Діаграма класів

Розроблений програмний продукт складається з чотирьох класів: клас Генератор рими (RhymeGenerator), клас Токен (TokenWeighter), клас Модель (Model) та клас Інструменти (Utils).

Клас Генератор рими (RhymeGenerator) містить такі атрибути:

– мовна модель для маскування токенів (model), що має тип класу Model;

– токенізатор (tokenizer), що має тип PreTrainedTokenizer;
– зважувач токенів (token_weighter), що має тип TokenWeighter.

Клас Генератор рими (RhymeGenerator) містить такі методи:

– почати (generateRhyme), що починає генерувати рядок;
– підбирає риму (initRhymes), повертає токенізований рядок;
– змінює риму (changeRhyme), що має тип рядка (string);
– маскує рядок і повертає індекс для оновлення (cretaeMaskForToken);
– оновлює маску чи рандомний токен (chooseTokenForMask);
– оновити дублюючі токени (repeatMasking);
– передбачити кандидати на місце масок(predictMaskedTokens);
– обчислити ймовірність і вагу токена (getReplacementInfo).

Клас Токен (TokenWeighter) містить такі атрибути:

– токенізатор (tokenizer), що має тип PreTrainedTokenizer;
– ймовірність токена (proba).
– Клас Токен (TokenWeighter) містить такі методи:
– обчислити ймовірність появи токена (getTokenProba);
– фільтр для підбору токенів (validToken).
– Клас Модель (Model) містить такі атрибути:
– кількість тестових даних (batch_size);
– розмірність вірша (word_type_size).
– Клас Модель (Model) містить такі методи:
– ініціалізувати процес створення вірша(initStructureModel);
– ініціалізувати процес створення рими (initRhymeModel);
– розрахувати помилку результатів (computeModelLoss);
– створити вірш (generate).

Клас Інструменти (Utils), який зберігає настройки для проекту і допоміжні функції:

– відобразити результат (showResult);
– завантажити модель (load_model);
– поставити маркер на нове слово (markWord);
– визначити останнє слово у рядку (getLastWord);
– запустити процес створення вірша (main).

Опишемо зв'язки між класами. Клас Модель (Model) та клас Генератор рими (RhymeGenerator) поєднані n-арною асоціацією, тобто на основі однієї моделі може бути створено багато генераторів рим. Клас Модель (Model) та клас Токен (TokenWeighter) поєднані n-арною асоціацією, тобто для однієї моделі може створювати безліч токенів. Клас Модель (Model) та клас Інструменти (Utils) поєднані n-арною асоціацією, тобто для однієї моделі може створювати безліч насройок/інструментів.

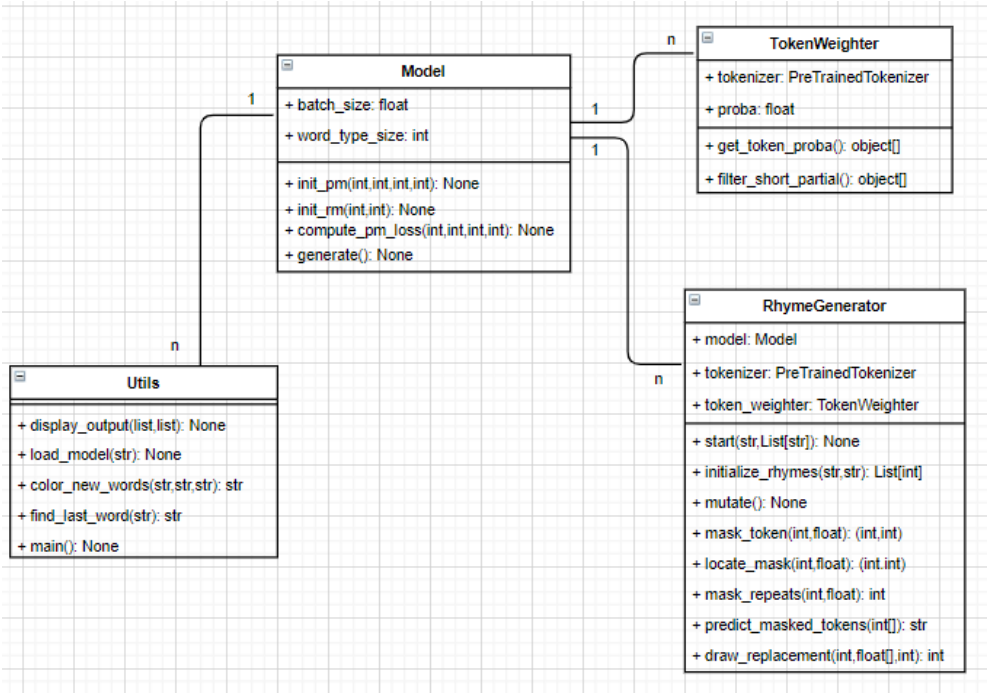


Рисунок 4.2 – Діаграма класів

4.3.2 Діаграма послідовності

Розглянемо структурну схему послідовності. Користувач має можливість ввести рядок для написання вірша. Для цього необхідно ввести рядок вірша у відповідне поле. У разі введення валідного рядка, запускається цикл функцій, які визиваються з main(). Починається пібір слів для продовження вірша, починаючи з кінця кожного рядка, для цього спочатку запускається підбір рими за допомогою функції initRhymes (). Після того, як рими були визначені 10% слів у рядках замінюються на маски за допомогою функції cretaeMaskForToken (). Для покращення результатів випадкові слова будуть маскуватися за допомогою функції repeatMasking (). Відповідно цей процес триває доки не будуть заповнені всі маски. Паралельно до цього результат процесу генерування вірша – підбору слів у вірш здійснюється за допомогою функції markWord (). Після закінчення процесу генерування вірша результат записується в базу даних і користувач може побачити результат та спробувати згенерувати вірш ще раз, ввівши новий рядок. Також при бажанні може згенерувати вірш ще раз, якщо результат не буде його задовільняти.

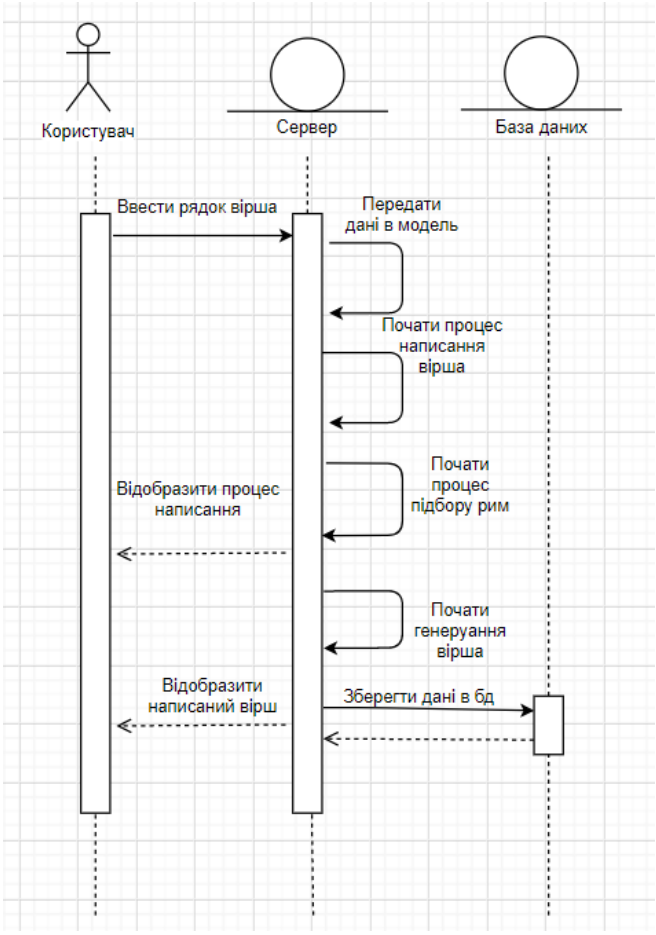


Рисунок 4.3 – Діаграма послідовності

4.3.3 Діаграма компонентів

Розглянемо структурну схему компонентів. З діаграми видно, що система містить такі компоненти: Генератор рими (RhymeGenerator), Модель (Model), Користувач (User), та База Даних (Database).

Компонент Генератор рими (RhymeGenerator) надає компоненту Модель (Model) необхідну інформацію про римування слів.

Компонент Модель (Model) надає компоненту Користувач (User) можливість створити вірш, враховуючи основні критерії для написання.

Компонент Користувач (User) надає компоненту Модель (Model) інформацію про вхідні дані, на основі яких потрібно генерувати вірш.

Компонент База Даних (Database) надає компоненту Модель (Model) можливість зберігати результати роботи моделі.

Рисунок 4.4 – Діаграма компонентів

4.3.4 Специфікація функцій

Опишемо функції класів програмного забезпечення. Детальний опис усіх функцій наведено у таблиці 4.2

Таблиця 4.2 – Специфікація функцій програмного забезпечення

Назва функції	Опис
generateRhyme ()	Функція класу RhymeGenerator (Генератор рими), що починає процес підбору рими.
initRhymes (string,string)	Функція класу RhymeGenerator (Генератор рими), що ініціалізує параметри процесу підбору рими.
changeRhyme ()	Функція класу RhymeGenerator (Генератор рими), що змінює риму при встановленні маски на римуюче слово.
cretaeMaskForToken (int,float)	Функція класу RhymeGenerator (Генератор рими), що маскує слово і повертає його індекс.
chooseTokenForMask (int,float)	Функція класу RhymeGenerator (Генератор рими), що оновлює маску чи випадковий токен.

Продовження таблиці 4.2

Назва функції	Опис
repeatMasking (int,float)	Функція класу RhymeGenerator (Генератор рими), що оновлює повторюючі токен.
predictMaskedTokens (ndarray)	Функція класу RhymeGenerator (Генератор рими), що обчислює ймовірність маскування токена.
getReplacementInfo (float,ndarray,int)	Функція класу RhymeGenerator (Генератор рими), що обчислює ймовірність, вагу і індекс токена.
getTokenProba ()	Функція класу TokenWeighter (Токен), що обчислює ймовірність появи токена.
validToken (dictionary)	Функція класу TokenWeighter (Токен), що фільтрує токени.
initStructureModel (int,int)	Функція класу Model (Модель), що настроює метр у вірші.
initRhymeModel (int,int)	Функція класу Model (Модель), що настроює риму у вірші.
computeModelLoss (int,int,int,int)	Функція класу Model (Модель), що вираховує помилку моделі.
generate()	Функція класу Model (Модель), що запускає процес створення вірша.
showResult (string,string,string,string)	Функція класу Інструменти (Utils), що відображає процес створення вірша.
load_model(string)	Функція класу Інструменти (Utils), що завантажує натреновану модель нейронної мережі.
markWord (string,string,string)	Функція класу Інструменти (Utils), що виділяє нове згенероване слово.
getLastWord (string)	Функція класу Інструменти (Utils), що визначає останнє слово у рядку.
main()	Функція класу Інструменти (Utils),запускає сервер.

4.4 Опис звітів

Під час тестування моделі нейронної мережі, обчислюється помилка підбору слів. Саме це значення дає інформацію про ефективність навчання моделі та її прогрес у передбаченні та генеруванні коректних слів. Іншим показником оцінювання якості роботи моделі – це якість згенерованого вірша. Якість вірша оцінюється по трьом основним критеріям: контекст,

рима і метр. Користувач перечитує вірші і ставить оцінки по п’ятибальній шкалі, після цього обчислюється середня оцінка для вірша і ми можемо оцінити якість вірша на певний момент часу. Розглянемо приклад оцінювання якості роботи моделі.

Нехай в процесі виконання тестування тестова вибірка віршів складається з трьох випадково вибраних віршів, що були оцінені по трьом критеріям. Тоді таблиця оцінок матиме вигляд, представлений у таблиці 4.3.

Таблиця 4.3 – Таблиця оцінок

	Контекст	Рима	Метр	Дата	Загальна оцінка
Вірш 1	4	4	4	05.05.2020	12
Вірш і	5	4	4	15.05.2020	13
Вірш n	5	5	4	27.05.2020	14
Загальна оцінка	14/15	13/15	12/15	-	39/45

Розглянемо першу колонку із таблиці 4.3. Видно, що всього було набрано 14 балів із 15. Це означає, що модель добре зберігає контекст – єдину тематику в усьому вірші. Рима зберігається на протязі всього вірша у кожній вибірці і загальна оцінка складає 13 балів. Це означає, що іноді генерувалися слова, які не вписувалися в контекст лиш би заповнити схему римування. Метр майже зберігається у всіх віршах тестової вибірки, але це є дуже хорошим результатом, адже модель підбирає слова, зберігаючи контекст, жертвуючи метром.

Для того щоб порахувати точність створеної моделі, необхідно просумувати всі оцінки поділити на максимальну кількість і помножити на 100%. У нашому випадку точність складатиме 87%.

Висновок до розділу

В даному розділі були розглянуті засоби розробки програмного забезпечення, наведено коротку характеристику засобів програмування, які були б корисними при вирішенні поставлених задач та досягненні мети. Також був обґрунтований вибір інструментів для програмування: мова програмування Python, платформа TensorFlow, Streamlit, а також бібліотек для роботи з текстом і базами даних, наприкладі PostgreSQL.

Крім цього були встановлені загальні вимоги до програмного забезпечення, описано бібліотеки і фреймворки, що використовуватимуться при розробці програмного продукту.

Була описана архітектурна модель програмного забезпечення. Спроектовано діаграми класів, послідовності та компонентів і наведено їх детальний опис. Також була наведена специфікація функцій, що використовувалися при створенні застосунку та описано звіт, за допомогою якого можна описати якість моделі.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Для того, щоб відкрити застосування на машині, де локально розгорнутий сервер, потрібно ввести в браузері http://localhost:8501/. Відкривається інтуєтивно-простий інтерфейс.

Користувач має можливість вввести рядок вірша і програма продовжить його, візуалізуючі процес підбору слів. Ця можливість представлена на головній і єдиній сторінці астосунку, що представлена на рисунку 5.1.

Рисунок 5.1 – Головна сторінка застосунку

Якщо користувач вводить рядок вірша і натискає клавішу «Enter», то навчена модель починає процес генерування вірша, де прогрес відображається у відповідній шкалі. Цей процес зображений на рисунку 5.2.

Рисунок 5.2 – Процес написання вірша

Після завершення процесу буде запущене анімація , яка означає кінець процесу, зображена на рисунку 5.3.

Рисунок 5.3 – Процес генерування віршів

Користувач може переривати процес генерації вірша, але і вводить новий рядок під час цього процесу і відповідно перезапускати процес. Цей процес зображений на рисунку 5.4.

Рисунок 5.4 – Переривання процесу генерування віршів

Після завершення генерації вірша, результат записується у базу даних як результат роботи моделі, результат наведений на рисунку 5.5.

Рисунок 5.5 – Результати роботи застосунку

Якщо користуач вводить некоректні дані чи модель не може підібрати коректне продовження, то вона виводить відповідне повідомлення, зображено на рисунку 5.6.

Рисунок 5.6 – Результати обробки некоректних даних

Користувач може ввести інші дані і перезапустити процес генерування.

5.2 Випробування програмного продукту

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій комплексу задач підтримки процесу написання віршів вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі таких документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;

– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

У процесі тестування була перевірена вся функціональність кмоплексу задач для підтримки процесу написання віршів. У таблиці 5.1 наведений перелік випробувань основних функціональних можливостей.

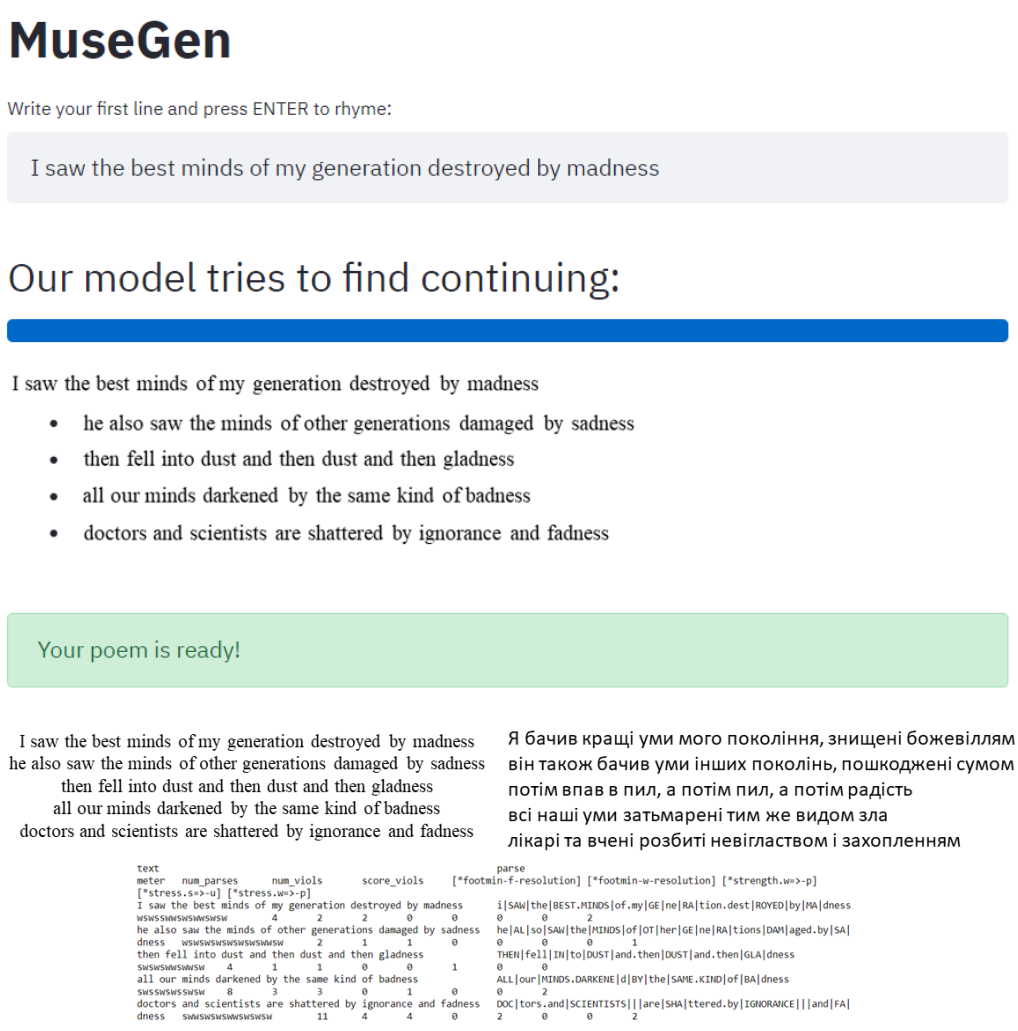
Таблиця 5.1 – Тест роботи моделі

Функція/ Use Case:	**Робота застосунку**	
	Очікуваний результат:	Результат тесту:
Інтерфейс користувача	Відкрита головна сторінка http://localhost:8501/. Всі елементи сторінки працюють	пройдений
Коректність введених вхідних даних	Виведення відповідного повідомлення для користувача	пройдений
Контекст у вірші	У вірші зберігається єдина мета/ідея вірша	пройдений
Рима у вірші	Останні слова рядків римуються між собою	пройдений
Метр у вірші	Зберігається чергування наголошених і ненаголошених складів у рядках	пройдений
Якість вірша за допомогою анкетування користувачів	Середня оцінка є більшою за чотири при максимальній п’ятірці	пройдений
Можливість перезапустити роботу моделі чи призупинити і завершити роботу	Меню для повного контроля процесу роботи моделі	пройдений

Тести були провеені вручну, моделюючи різні можливі ситуації. Всі тести були направлені на виявлення в застосунку невідповідностей заявленим характеристикам.

Всі тести були успішно пройдені.

Наведемо результати виконання програми.



I saw the best minds of my generation destroyed by madness
he also saw the minds of other generations damaged by sadness
then fell into dust and then dust and then gladness
all our minds darkened by the same kind of badness
doctors and scientists are shattered by ignorance and fadness

Я бачив кращі уми мого покоління, знищені божевіллям
він також бачив уми інших поколінь, пошкоджені сумом
потім впав в пил, а потім пил, а потім радість
всі наші уми затьмарені тим же видом зла
лікарі та вчені розбиті невіглаством і захопленням

text				parse			
meter	num_parses	num_viols	score_viols	[*footmin-f-resolution]	[*footmin-w-resolution]	[*strength.w=>-p]	[*stress.s=>-u] [*stress.w=>-p]
I saw the best minds of my generation destroyed by madness	wswsswswswswswsw	4	2	2	0	0	i|SAW|the|BEST.MINDS|of.my|GE|ne|RA|tion.dest|ROYED|by|MA|dness
he also saw the minds of other generations damaged by sadness	wswswswswswswswsw	2	1	1	0	0	0	1	he|AL|so|SAW|the|MINDS|of|OT|her|GE|ne|RA|tions|DAM|aged.by|SA|dness
then fell into dust and then dust and then gladness	swswswswswsw	4	1	1	0	0	1	0	0	THEN|fell|IN|to|DUST|and.then|DUST|and.then|GLA|dness
all our minds darkened by the same kind of badness	swsswswsswsw	8	3	3	0	1	0	0	2	ALL|our|MINDS.DARKENE|d|BY|the|SAME.KIND|of|BA|dness
doctors and scientists are shattered by ignorance and fadness	swswswswswswswsw	11	4	4	0	2	0	0	2	DOC|tors.and|SCIENTISTS|||are|SHA|ttered.by|IGNORANCE|||and|FA|dness

Рисунок 5.7 – Результати роботи програми

Someday you will cry for me, but not today
for today you will be giving me my special day
today you cry for the people in light and grey
look what happened and now you will have to pay
Ashe thought as he watched as his father walked away

Коли-небудь ти будеш плакати за мене, але не сьогодні
на сьогодні ти даси мені мій особливий день
сьогодні ти плачеш за людей в світлих і сірих тонах
подивися що трапилося і тепер тобі доведеться заплатити
Подумав Еш, дивлячись, як його батько йде

text				parse			
meter	num_parses	num_viols	score_viols	[*footmin-f-resolution]	[*footmin-w-resolution]	[*strength.w=>-p]	[*stress.s=>-u] [*stress.w=>-p]
Someday you will cry for me, but not today	sswswswswsws	3	2	2	0	1	0	SOM.EDAY|you.will|CRY|for|ME|but|NOT|to|DAY	1 0
for today you will be giving me my special day	swswswswswsws	10	3	3	0	2	0	FOR|to|DAY|you.will|BE.GI|ving|ME|my|SPE|cial|DAY	0 1
today you cry for the people in light and grey	wswswswswsws	5	1	1	0	0	0	to|DAY|you|CRY|for.the|PEO|ple.in|LIGHT|and|GREY	0 1
look what happened and now you will have to pay	swswswswsws	2	1	1	0	0	0	LOOK|what|HAP|pened.and|NOW|you.will|HAVE|to|PAY	0 1
Ashe thought as he watched as his father walked away	sswswswswsws	2	1	1	0	0	0	ASHE.THOUGHT|as.he|WATCHED|as.his|FAT|her|WALKED|a|WAY	0 1

Рисунок 5.8 – Результати роботи програми

Таблиця 5.1 – Тест роботи моделі

Оригінал	Переклад
I saw the best minds of my generation destroyed by madness he also saw the minds of other generations damaged by sadness then fell into dust and then dust and then gladness all our minds darkened by the same kind of badness doctors and scientists are shattered by ignorance and fadness	Я бачив кращі уми мого покоління, знищені божевіллям. він також бачив уми інших поколінь, пошкоджені сумом. потім впав у пил, а потім пил, а потім радість, всі наші уми затьмарені тим же видом зла. лікарі та вчені розбиті невіглаством і захопленням.
Someday you will cry for me, but not today for today you will be giving me my special day today you cry for the people in light and grey look what happened and now you will have to pay Ashe thought as he watched as his father walked away	Коли-небудь ти будеш плакати за мене, але не сьогодні. на сьогодні ти даси мені мій особливий день. сьогодні ти плачеш за людей в світлих і сірих тонах. подивися що трапилося і тепер тобі доведеться заплатити - Подумав Еш, дивлячись, як його батько йде
In the dark the sun gleam, And in the dark the moon seem But now the evening is begun— And has set on the earth the sun!	У темряві світить сонце, І в темряві воно здається місяцем Але зараз розпочався вечір - І сонце сіло на землю!
Death, be not proud, though some have called same Mighty and dreadful, but you are not so; For those whom you think can overthrow And you can not kill me.	Смерть, не гордися собою, хоча деякі називали тебе могутньою та жахливою. Але ти не така для тих, кого ти думаєш, що зможеш знищити. І ти не можеш вбити мене.
The trees are in their autumn beauty, The woodland paths are dry. Under the October twilight The water mirrors a still sky.	Дерева у своїй осінній красі, Сухі лісові стежки. У жовтневих сутінках Вода відзеркалює нерухоме небо.

Продовження таблиця 5.1

Оригінал	Переклад
When I am dead, my dearest, Sing no sad songs for me; Plant no roses at my head, Nor shady cypress tree: Be the green grass above me And if you will, remember, And if not - forget.	Коли я помру, мою дорога, Не співай для мене сумних пісень; Не сади над моєю головою ні троянд, Ні тінистих кипарис: Нехай буде зелена трава наді мною А якщо ти хочеш, пам’ятай, А якщо ні, то забудь.
The larger streams run still and deep, Noisy and fast the small brooks run Among the mullein stalks the sheep Go up the hillside in the sun	Великі потоки течуть тихо и глибоко, Шумно и швидко біжать маленькі струмочки. Між кленових гілок вівці Піднімаються на холм на сонце

Згенеровано сім віршів, оцінимо їх якість по критеріях, зазначених у таблиці 5.2.

Таблиця 5.2 – Тест роботи моделі

Рима	Ритм	Контекст
7/7	5/7	7/7

Слід зазначити, що згенеровані вірші можуть потребувати додаткового доопрацювання, але незначного. Окрім того, зі збільшенням віршів у базі даних, якість згенерованих віршів підвищиться.

У подальшому планується додати можливість вибору потрібних схем римування, що також підвищить якість отриманих віршів.

Висновок до розділу

У даному розділі дипломного проекту, було визначено, сформовано і описано керівництво користувача та наведені приклади скріншотів розробленого застосунку і їх детальний опис. Детально був описаний процес роботи користувача із застосунком та можливі сценарії його роботи.

Також була встановлена мета проведення випробувань, наведені загальні положення. Були визначені і описані тести, які необхідно провести

для максимального покриття роботи застосунку і первірці відповідності зазначеним вимогам.

Було проведене тестування, у процесі якого була перевірена функціональність застосунку. У ході тестування програмного продукту було встановлено, що функціонал розробленого застосунку відповідає встановленим вимогам, тобто усі тестові сценарії були успішно оброблені системою.

Для кращого вирішення спеціальних задач модель потрибує додаткового коригування і донавчання.

ЗАГАЛЬНІ ВИСНОВКИ

У ході написання даного розділу дипломного проекту було описано предметне середовище, яке розглядається. Було показано, що люди ще не досягли значних успіхів при розв'язанні задач пов'язаних із генеруванням віршів, а зростання кількості нових алгоритмів та одночасне зростання доступної обчислювальної потужності комп'ютерів дозволяють покращити процес написання віршів.

Окрім цього було описано процес діяльності, який покращується, визначено як покроково розв'язати поставлену задачу, наведено коротку характеристику підходу, яким пропонується вирішувати поставлену задачу. Було побудовано діаграму діяльності та наведено її опис.

Також був описаний функціонал системи та визначено функції користувача. Був проведений аналіз наявних існуючих аналогів та визначення відмінностей, які матиме програмний продукт, що розробляється.

Сформовано призначення розробки, встановлено мету та визначено задачі, які необхідно вирішити для досягнення мети.

У ході написання були визначені вхідні та вихідні дані системи, що розробляється.

Надано детальний опис вхідних даних, що задаються розробником і користувачем.

Встановлені параметри за замовчуванням для моделі нейронної мережі. Користувач при наявності коду може їх змінити і перевчити чи довчити мережу.

Також, проаналізовано важливість використання бази даних для збереження необхідної інформації та подальший аналіз якості створених віршів. Спроектовано модель бази даних та її структуру. Наведено детальний опис атрибутів таблиці бази даних, визначено їх тип.

Під час написання дипломного проекту було наведено змістовну та математичну постановки задачі, визначено основні терміни, які будуть зустрічатися при описі поставлених задач.

В змістовній постановці задачі був наведений приклад, який допоможе детальніше ознайомитися із задачами та можливими методами машинного навчання для їх вирішення.

В математичній постановці задач були сформульовані задачі та описані їх основні атрибути. Визначено математичний запис кожної із задач і глобальної – загальної задачі відповідно.

Також, було наведено опис методів розв’язання задачі, наведено характеристику кожного із методів. Встановлено переваги та недоліки кожного із підходів.

Крім цього було наведено математичне обґрунтування кожного із підходів. Встановлено і доведено, правильність використання обраного методу машиного навчання для вирішення поставлених задач.

Задача, що була представлена у змістовній постановці була розв’язана одним із методів машинного навчання, який стає популярним із кожним роком у роботі з природньою мовою.

Також були розглянуті засоби розробки програмного забезпечення, наведено коротку характеристику засобів програмування, які були б корисними при вирішенні поставлених задач та досягненні мети. Також був обґрунтований вибір інструментів для програмування: мова програмування Python, платформа TensorFlow, Streamlit, а також бібліотек для роботи з текстом і базами даних, наприкладі PostgreSQL.

Крім цього були встановлені загальні вимоги до програмного забезпечення, описано бібліотеки і фреймворки, що використовуватимуться при розробці програмного продукту.

Була описана архітектурна модель програмного забезпечення. Спроектовано діаграми класів, послідовності та компонентів і наведено їх детальний опис. Також була наведена специфікація функцій, що використовувалися при створенні застосунку та описано звіт, за допомогою якого можна описати якість моделі.

Було визначено, сформовано і описано керівництво користувача та наведені приклади скріншотів розробленого застосунку і їх детальний опис. Детально був описаний процес роботи користувача із застосунком та можливі сценарії його роботи.

Також була встановлена мета проведення випробувань, наведені загальні положення. Були визначені і описані тести, які необхідно провести для максимального покриття роботи застосунку і первірці відповідності зазначеним вимогам.

Було проведене тестування, у процесі якого була перевірена функціональність застосунку. У ході тестування програмного продукту було встановлено, що функціонал розробленого застосунку відповідає встановленим вимогам, тобто усі тестові сценарії були успішно оброблені системою.

Слід також зазначити, що для кращого вирішення спеціальних задач (підвищення якості підбору рими, дотримання метроритму вірша та його контексту) модель потребує додаткового корегування і донавчання за рахунок збільшення датасету, додавання можливості вибору типу римування, тощо.

ПЕРЕЛІК ПОСИЛАНЬ

1. Хоменко О.М., Гавриленко О.В. Використання нейронних мереж для написання віршів. Інформаційні системи та технології управління. 2019. №3, С. 222–230.

2. Хоменко О.М., Гавриленко О.В. Інформаційна система з підтримки написання віршів. Інформаційні системи та технології управління. 2020. №4., С. 118–121.

3. An introduction to Data Science [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@onejohi/an-introduction-to-data-science-32403b22f5c1.

4. Your Guide to Natural Language Processing (NLP). How machines process and understand human language [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1.

5. Deep Learning for NLP: An Overview of Recent Trends [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/dair-ai/deep-learning-for-nlp-an-overview-of-recent-trends-d0d8f40a776d.

6. Types of Machine Learning Algorithms You Should Know [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861.

7. 14 Different Types of Learning in Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/types-of-learning-in-machine-learning/.

8. Recurrent Neural Networks Remembering what's [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce.

9. What is a Transformer? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04.

10. Understanding BERT architecture [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/understanding-bert-architecture-3f35a264b187.

11. OpenAI GPT2 [Електронний ресурс] – Режим доступу до ресурсу: https://huggingface.co/transformers/model_doc/gpt2.html.

12. Introduction to Hidden Markov Models [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/introduction-to-hidden-markov-models-cd2c93e6b781.

13. Hidden Markov Models [Електронний ресурс] – Режим доступу до ресурсу: https://web.stanford.edu/~jurafsky/slp3/A.pdf.

14. Hidden Markov Models and their Applications in Biological Sequence Analysis [Електронний ресурс] – Режим доступу до ресурсу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2766791/.

15. Hidden Markov Model [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@kangeugine/hidden-markov-model-7681c22f5b9.

16. RNN-LSTM-GRU based language transformation [Електронний ресурс] – Режим доступу до ресурсу: https://link.springer.com/article/10.1007/s00500-019-04281-z.

17. Visualizing memorization in RNNs [Електронний ресурс] – Режим доступу до ресурсу: https://distill.pub/2019/memorization-in-rnns/.

18. Why does the transformer do better than RNN and LSTM in long-range context dependencies? [Електронний ресурс] – Режим доступу до ресурсу: https://ai.stackexchange.com/questions/20075/why-does-the-transformer-do-better-than-rnn-and-lstm-in-long-range-context-depen.

19. Comparing the Performance of the LSTM and HMM Language Models via Structural Similarity [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/pdf/1907.04670.pdf.

20. Recurrent Neural Networks cheatsheet [Електронний ресурс] – Режим доступу до ресурсу: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

21. Transformer Neural Network Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://devopedia.org/transformer-neural-network-architecture.

22. Visualizing machine learning one concept at a time [Електронний ресурс] – Режим доступу до ресурсу: http://jalammar.github.io/illustrated-transformer/.

23. What is a Transformer? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04.

24. Attention Is All You Need [Електронний ресурс] – Режим доступу до ресурсу: https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

25. What is PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: postgresql.org/about/.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО
Керівник проєкту
_____ *Олена ГАВРИЛЕНКО*
(підпис) (вл. ім’я, прізвище)
“13” квітня 2020 р.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)
“14” квітня 2020 р.

Комплекс задач підтримки процесу написання віршів

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 6128.01.000 ТЗ*

на 8 сторінках

Київ – 2020 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ…….. 3

1.1 Повне найменування системи та її умовне позначення…….. 3

1.2 Найменування організації-замовника та організацій-учасників робіт…… 3

1.3 Перелік документів, на підставі яких створюється система (Завдання на ДП)……... 3

1.4 Планові терміни початку і закінчення роботи зі створення системи……... 4

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ КОМПЛЕКСУ ЗАДАЧ……..... 4

2.1 Призначення комплексу задач……... 4

2.2 Цілі створення комплексу задач..…….. 4

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ…… 5

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ……..... 6

4.1 Вимоги до функціональних характеристик…….. 6

4.2 Вимоги до надійності…….. 6

4.3 Умови експлуатації (тільки для систем, специфіка яких передбачає особливі умови експлуатації)……. 6

4.4 Вимоги до складу і параметрів технічних засобів…… 6

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ……. 7

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ……. 8

6.1 Види випробувань…… 8

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повна назва системи: Комплекс задач підтримки процесу написання віршів.

Коротке найменування системи: «MuseGen».

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовником є кафедра автоматизованих систем обробки інформації та управління Національного технічного університету України "Київський політехнічний інститут ім. Ігоря Сікорського" (далі за текстом — Замовник). Адреса замовника: м. Київ, п. Перемоги 37, 18 корпус ФІОТ.

Розробник сервісу — студент групи ІС-61 кафедри автоматизованих систем обробки інформації та управління Національного технічного університету України "Київський політехнічний інститут ім. Ігоря Сікорського" Хоменко Олександр Миколайович.

1.3 Перелік документів, на підставі яких створюється система

При розробці системи і створення проектно-експлуатаційної документації Виконавець повинен керуватися вимогами наступних нормативних документів:

– ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;

– ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи;

– ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплексність і позначення документів при створенні автоматизованих систем.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий строк початку роботи по розробці «MuseGen» — 17 лютого 2020 року.

Плановий строк закінчення роботи по розробці «MuseGen» —25 травня 2020 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ КОМПЛЕКСУ ЗАДАЧ

2.1 Призначення комплексу задач

Веб-застосунок призначений для полегшення процесу написання, створення віршів і підборі рим в режимі онлайн.

2.2 Цілі створення комплексу задач

Цілями розробки системи є:

- прискорення процесу створення віршів;
- покращити якість написання віршів;
- покращити підбір рим.

Для досягнення поставлених цілей необхідно реалізувати наступні задачі. Задачами розробки є:

- зберегти єдиний контекст у вірші;
- створення римування;
- контролювання метрики вірша.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Для користування застосуванням, користувач повинен ввести перший рядок вірша.

Об’єктом автоматизації є сам процес створення віршів.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Застосунок має здійснювати генерування віршів для користувачів: можуть створювати вірші відповідно до введеного рядків.

Застосунок має виконувати наступні функції:

- створення віршів;
- візуалізація процесу генерування вірша.

4.2 Вимоги до надійності

Сервіс повинен безвідмовно функціонувати. Виправлення виявлених проблем повинно здійснюватися при тестуванні.

4.4 Вимоги до складу і параметрів технічних засобів

Для коректної роботи даного продукту на стороні користувача мають бути такі технічні характеристики:

1. комп’ютер:
   - процесор з частотою - не менше 2 ГГц;
   - об’єм оперативної пам’яті - не менше 4 Гб.
2. програмне забезпечення:
   - операційна система Windows 10;
   - браузер.
3. комп’ютерна периферія:
   - монітор;
   - мишка;
   - клавіатура.

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Основні етапи виконання робіт з розробки системи ведення наукової роботи.

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Визначення цілей та задач	12.02.2020	
2.	Розробка сценарію роботи програми	15.02.2020	
3.	Узгодження з керівником теоретичної частини	21.02.2020	
4.	Узгодження з керівником інтерфейсу користувача	21.02.2020	
5.	Розробка програмного забезпечення	17.03.2020	
6.	Тестування програми	30.03.2020	
7.	Покращення програми	09.04.2020	
8.	Створення документації	21.04.2020	
9.	Здача готового продукту замовнику	22.05.2020	

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

6.1 Види випробувань

Для контролю правильності роботи програмного забезпечення буде проведено функціональне тестування. В ході тестування буде проведено випробування основних функціональних характеристик застосунку та цілої моделі загалом.

Тестування згенерованих віршів полягає у перевірці рими, метрики та контексту.

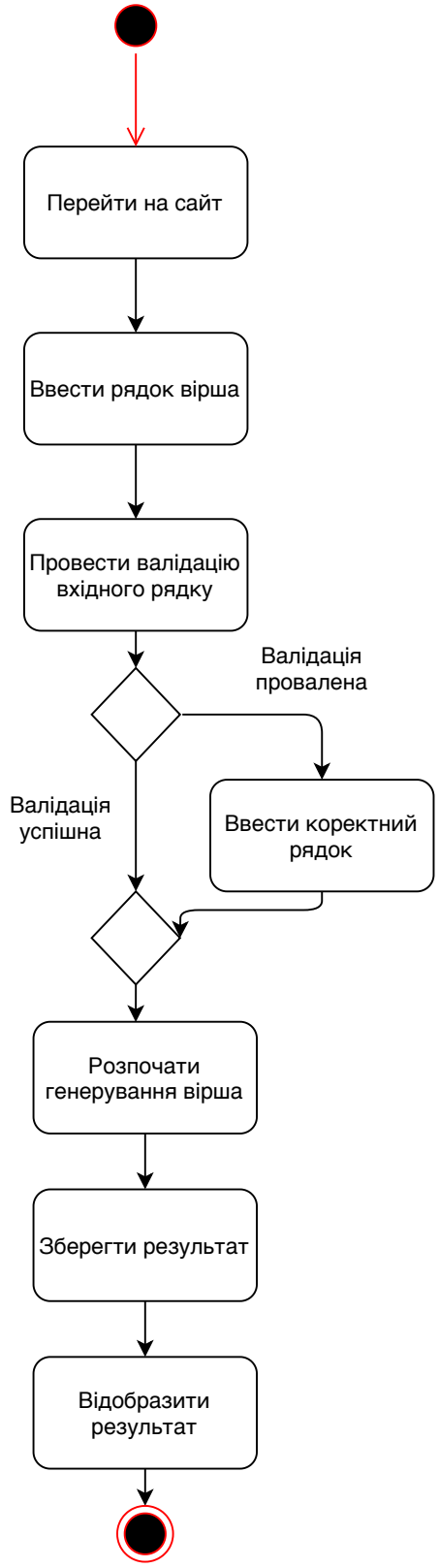
Графічний матеріал
до дипломного проєкту

на тему: Комплекс задач підтримки процесу написання віршів

Київ – 2020 року

					ДП ІС-6128.XXXX-с.ССК	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів	Літера / Маса / Масштаб
Розробив		Хоменко О.М				
						Аркуш 1 / Аркушів 1
Перевірив		Гавриленко О.В				
Т. кон.					Комплекс задач підтримки процесу написання віршів	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61
Н. кон.		Тєлишева Т.О				
Затвердив		Гавриленко О.В				

					ДП ІС-6128.XXXX-с.ССК	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів	Літера / Маса / Масштаб
Розробив		Хоменко О.М				
Перевірив		Гавриленко О.В				Аркуш 1 / Аркушів 1
Т. кон.					Комплекс задач підтримки процесу написання віршів	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61
Н. кон.		Тєлишева Т.О				
Затвердив		Гавриленко О.В				



					ДП ІС-6128.ХХХХ-с.ССД	
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна діяльності процесу роботи застосунку*	Літера / Маса / Масштаб
Розробив		Хоменко О.М				
						Аркуш 1 / Аркушів 1
Перевірив		Гавриленко О.В				
Т. кон.					*Комплекс задач підтримки процесу написання віршів*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61*
Н. кон.		Тєлишева Т.О				
Затвердив		Гавриленко О.В				

					ДП ІС-6128.ХХХХ-с.ССП	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна послідовності	Літера / Маса / Масштаб
Розробив		Хоменко О.М				
Перевірив		Гавриленко О.В				Аркуш 1 / Аркушів 1
Т. кон.					Комплекс задач підтримки процесу написання віршів	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61
Н. кон.		Тєлишева Т.О				
Затвердив		Гавриленко О.В				

					ДП ІС-6128.ХХХХ-с.ССВ		
						Літера	Маса Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використання		
Розробив		Хоменко О.М				Аркуш 1	Аркушів 1
Перевірив		Гавриленко О.В					
Т. кон.					Комплекс задач підтримки процесу написання віршів	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61	
Н. кон.		Тєлишева Т.О					
Затвердив		Гавриленко О.В					